UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Ilê Caian Gums

IMPLEMENTAÇÃO E ANÁLISE DE UMA BIBLIOTECA DE SEGREDO COMPARTILHADO

Florianópolis

2018

Ilê Caian Gums

IMPLEMENTAÇÃO E ANÁLISE DE UMA BIBLIOTECA DE SEGREDO COMPARTILHADO

Trabalho de Conclusão de Curso submetido ao Departamento de Informática e Estatística para a obtenção do Grau de Bacharel em Ciência da Computação.
Orientador: M.e Rick Lopes de Souza
Coorientador: Prof. Dr. Ricardo Felipe Custódio

Florianópolis

2018

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.

Tamanho: 7cm x 12 cm

Fonte: Times New Roman 9,5

Maiores informações em:
http://www.bu.ufsc.br/design/Catalogacao.html

Ilê Caian Gums

IMPLEMENTAÇÃO E ANÁLISE DE UMA BIBLIOTECA DE SEGREDO COMPARTILHADO

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Ciência da Computação", e aprovado em sua forma final pelo Departamento de Informática e Estatística.

Florianópolis, 26 de Novembro 2018.

Prof. Dr. Rafael Cancian
Coordenador do Curso

Prof. Dr. Ricardo Felipe Custódio
Coorientador

Banca Examinadora:

Prof. Dr. Ricardo Felipe Custódio
Presidente

M.e Rick Lopes de Souza
Orientador

Prof. Dr. Martín Augusto Gagliotti Vigil

AGRADECIMENTOS

Por me concederem o direito a vida e por sacrifícios diversos a mim e a minha irmã Luana e durante essa jornada que se concretiza hoje, agradeço de forma incondicional a meus pais: Aldo e Cilene. Agradeço também a minha irmã Luana, sempre presente em momentos importantes e agora segue seu próprio caminho ao lado de seu companheiro Artur, a quem também agradeço por trazer alegria e felicidade a nossa família. Transfiro aqui também meus agradecimentos a todos os outros membros de minha família que mesmo distantes sempre são lembrados em diversos momentos.

Agradeço a todo o grupo que me acompanhou em minha jornada na UFSC e no LabSEC, principalmente a todos aqueles que tive um maior contato. Em especial ao Professor Custódio que com sua história sempre me motivou a seguir em frente sem desistir, ao Rick que aceitou de prontidão essa tarefa de me orientar também em meu Trabalho de Conclusão e ao Professor Martín que mesmo com o pouco contato que tive também foi muito receptivo e acessível. A todos que já fizeram parte do LabSEC e aqueles que ainda fazem: obrigado pelas conversas, risadas e lições aprendidas. Não poderia citar todos sem esquecer de algum de vocês mas é certo que fizeram parte de momentos inesquecíveis a mim.

Agradeço a todos aqueles que me envolvi durante a Graduação e aqueles que ainda tenho contato. Em especial ao fiel amigo de caminhadas anteriores Luis e nossos cafés filosóficos sobre temas variados, ao Felipe e seu irmão Everton que sempre tiveram motivação para ouvir os diversos temas que eu sempre trouxe a nossa convivência. A todos aqueles que não mencionei, meu muito obrigado por sempre estarem ao meu lado durante essa época de minha vida.

Para aqueles que não estiveram comigo diretamente relacionados a graduação mas que participaram de minha vida durante essa época, deixo aqui também um grandioso agradecimento. Em especial ao Endy que sempre me acompanhou e não pestanejou ao me ajudar em um momento tão importante de sua vida (e a sua Noiva Andressa por permitir essa ajuda) e ao Rodolfo que mesmo tendo cruzado caminho comigo em minha época Labseciana tive um contato maior mais recente

e sempre me motiva com sua força de vontade.

A todos os que aqui não foram mencionados mas existem em minha memória de uma forma ou de outra, levo um pouco de sua sabedoria e de seus ensinamentos comigo. Obrigado por tudo e espero um dia poder retribuir esse sentimento de gratidão da melhor forma possível.

Por último mas não menos importante agradeço a Jéssica, que me acompanhou de perto durante toda essa jornada sempre trazendo conforto nos momentos difíceis, partilha de minhas alegrias nos momentos felizes e tem o dom da paciência para comigo de modo messiânico. Desejo que sua vida transborde de realizações, alegrias e sucesso e espero ter a honra de estar ao seu lado participando de tudo.

Bilbo: "Can you promise that I'll come back?"
Gandalf: "No. And if you do, you will not be the same."

J.R.R. Tolkien

SUMÁRIO

1 INTRODUÇÃO 11
1.1 OBJETIVOS 13
1.1.1 Objetivo Geral 13
1.1.2 Objetivos Específicos 13
1.2 JUSTIFICATIVA 13
1.3 MOTIVAÇÃO 14
1.4 METODOLOGIA 14
1.5 LIMITAÇÕES DO TRABALHO 15
1.6 ORGANIZAÇÃO DO TRABALHO 15
2 FUNDAMENTAÇÃO TEÓRICA 17
2.1 PRIVACIDADE 17
2.2 CONCEITOS BÁSICOS 18
2.3 CRIPTOGRAFIA 20
2.4 ALGORITMOS CRIPTOGRÁFICOS SIMÉTRICOS 21
2.5 ALGORITMOS CRIPTOGRÁFICOS ASSIMÉTRICOS 23
2.6 ASSINATURA DIGITAL 24
2.7 ASSINATURA DIGITAL E PROTOCOLOS DE AUTENTICAÇÃO 24
2.8 GERENCIAMENTO DE CHAVES 27
2.9 SEGREDO COMPARTILHADO 28
2.9.1 Definição 30
2.9.2 Esquema de Blakley 30
2.9.3 Esquema de Shamir 33
2.10 TRABALHOS RELACIONADOS 37
2.10.1 libgfshare 38
2.10.2 python-gfshare 43
2.10.3 Biblioteca sss 43
3 PROPOSTA 47
3.1 AMBIENTE E FERRAMENTAS 47
3.2 ORGANIZAÇÃO DE DIRETÓRIOS 48
3.3 ARQUITETURA 50
3.3.1 Formato de Arquivo `.share` 55
3.4 IMPLEMENTAÇÃO 57
3.4.1 Namespace 59
3.4.2 Escolhas e Padrões de Projeto 60
3.4.3 Diagrama de Classes 61
3.4.4 Classe `Controller` 62

3.4.5 Classe `ShamirDealer` 69
3.4.6 Classe `FileHandler` 74
3.4.7 Testes 76
3.5 UTILIZAÇÃO E EXEMPLOS 78
3.5.1 Executável *Standalone* 80
4 ANÁLISE 85
4.1 MODELAGEM 85
4.2 SEGURANÇA 96
5 CONCLUSÃO 107
5.1 TRABALHOS FUTUROS 108
ANEXO A – Artigo 113
ANEXO B – Código 119
REFERÊNCIAS 165

1 INTRODUÇÃO

A necessidade de proteção a dados confidenciais sempre foi uma constante em diversos momentos da história. O conhecimento acerca de conceitos como Criptografia, Segurança e Sigilo dos Dados e Comunicação Segura estão se tornando temas recorrentes tanto em ambientes distantes da academia quanto em estudos aprofundados com o estado da arte da Segurança em Computação. Teoria e prática têm avançado. No campo teórico, diversos conceitos matemáticos e protocolos de segurança tem sido publicados. No campo prático, as aplicações baseadas nessas publicações se aproximam cada vez mais da realidade palpável. Entre essa relação, podemos citar uso de Criptografia de Chaves Simétrica e Assimétrica na troca de mensagens eletrônicas bem como o uso de um Certificado Digital para a Assinatura Digital de um documento tornando este mesmo documento válido e com o mesmo poder jurídico de uma assinatura feita a punho.

Separados, os protocolos e conceitos matemáticos podem não ser suficientes para garantir aplicações cumprindo suas tarefas com relação as garantias necessárias exigidas pelo usuário final ou o cliente. Analisando o tempo de utilização de um Certificado Digital na hora de emissão (um dia, uma semana, um ano), a necessidade de se informar o tempo pelo qual ele permanecerá válido é essencial para o bom funcionamento de um protocolo de troca de mensagens, por exemplo. Quando este tempo chega ao fim o Certificado deve ser considerado obsoleto e a geração de um Certificado deve ser considerada para que a troca de mensagens permaneça segura. Realizar esse processo é essencial na continuidade de todo um Protocolo e no uso de Políticas, entretanto a busca por novas formas de se resolver esse mesmo problema ou outros não mencionados deve ser sempre incentivada. A busca pela resolução dos problemas e pela otimização de descobertas já realizadas é e sempre foi um importante combustível aos pesquisadores de todas as áreas e com o campo da Segurança em Computação não foi diferente.

Na análise de um problema caracterizado pela disponibilização de Chaves Criptográficas a todas as partes envolvidas de maneira segura, foi proposto por Shamir um protocolo de compartilhamento de dados que tinha como função a distribuição de um dado as partes envolvidas de maneira segura e o uso do dado distribuído pelas partes de maneira conjunta em algum tipo de combinação. Esse protocolo foi nomeado Segredo Compartilhado.

Shamir apresenta uma maneira de compartilhar um determinado

dado por suas partes envolvidas sem o conhecimento do dado em sua totalidade pelas partes e o uso desse dado restrito ao uso apenas com um determinado conjunto de partes reunidas no momento do uso. É proposta a realização desse procedimento com o uso de um polinômio mas o conceito apresentado pode ser generalizado e estendido.

Se deseja a divisão de um determinado dado de maneira segura para o acesso a ele em um grupo de n partes (pessoas, servidores, sistemas embarcados, etc). Ainda assim, queremos que isso seja feito de tal forma que se existem presentes no momento desejado de uso **pelo menos** t partes das n partes de tal forma que $1 < t < n$, é possível realizar o acesso a este dado. Também não é importante quais partes estejam presentes, com a garantia de que caso um número menor de t partes esteja presente nenhuma informação pode ser obtida sobre o dado.

Desde sua publicação em 1979, ainda não é possível encontrar com facilidade aplicações que utilizem ou disponibilizem o uso desse conceito em forma de ferramenta entre suas funcionalidades. A dificuldade em encontrar implementações para o uso em larga escala da funcionalidade de Segredo Compartilhado e que se distancie de uma mera forma de prova de conceito é a realidade. As formas de implementação das provas de conceito no que dizem respeito a Linguagens de Programação se limitam as linguagens consolidadas como `C/C++` ou `Java` e carecem de documentação. Não é possível a visualização ou teste do código produzido sendo possível apenas o uso de um executável onde não se tem a certeza de que o processo foi implementado de maneira correta, se existem ou não, erros de implementação ou ainda se garantem que estão livres de qualquer tipo de ataque durante o uso do executável.

Algumas análises foram realizadas em busca de alternativas para o trabalho proposto e como primeira implementação analisada foi encontrada **libgfshare**, que é implementada sobre o Corpo de *Galois* GF(2**8) que está disponível no repositório oficial do Sistema Operacional Ubuntu e está atualmente em sua versão 2.0. Em sequência é apresentada uma implementação em `Python` que encapsula a **libgfshare** apenas em forma de contextualização.

Como próxima implementação analisada temos a biblioteca **ssslib** (**??**), publicada por Daan Sprenkels em seu github para a linguagem *C* e com portes para outras plataformas como *NodeJS* e *Golang*, possuíndo uma análise em seu arquivo de `Readme.md`.

Com o intuito da preencher uma carência constatada, o trabalho apresentado tem como proposta a produção de uma biblioteca de

Segredo Compartilhado aberta e livre para o uso em larga escala com garantias da correta execução do protocolo e com possibilidade de verificação por terceiros dessas garantias.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Visando preencher a carência atual em uma implementação confiável, segura e expansível de Segredo Compartilhado, este trabalho possui o objetivo de desenvolver e disponibilizar de maneira aberta e livre de uma implementação em `C++` de Segredo Compartilhado.

1.1.2 Objetivos Específicos

- Desenvolver uma arquitetura de biblioteca que contemple escalabilidade.
- Implementar uma biblioteca de Segredo Compartilhado em C++ com disponibilidade para os Sistemas Operacionais baseados em Linux de forma que possa ser incluída em projetos futuros ou ser usada como um executável de maneira *stand-alone*.
- Realizar testes durante o desenvolvimento como auxílio a prevenção dos principais erros no desenvolvimento como vazamentos de memória(*memory leaks*) e afins.
- Realizar testes da biblioteca após seu desenvolvimento garantindo as propriedades necessárias para o correto funcionamento segundo a proposta original
- Produzir uma documentação para o auxílio ao uso da biblioteca desenvolvida.

1.2 JUSTIFICATIVA

Atrás de uma biblioteca que realizasse as operações propostas por Shamir em seu trabalho e que resolvessem o problema de compartilhamento e uso conjunto de um dado secreto sempre foi uma discussão recorrente e a abordagem adotada comum costumava ser o uso de uma

antiga, única e nada documentada biblioteca que permeava vários projetos dentro do LabSEC.

Discussões acerca da implementação de uma ferramenta interna pelos participantes da comunidade acadêmica da UFSC, em específico dos membros do LabSEC sempre foi considerada mas a existência de uma ferramenta que foi outras vezes utilizada era sempre um fator determinante no futuro dos projetos e apesar de buscas por alternativas acontecerem a escolha do uso da mesma ferramenta era o caminho tomado.

1.3 MOTIVAÇÃO

Em busca da elucidação e de uma garantia ainda maior no produto final de comunidades como o LabSEC e da eventual existência de soluções corporativas fechadas ao problema relativo a inexistência de uma ferramenta aberta e livre com garantias, testes, escalabilidade e documentação, é de desejo principal a disponibilização do resultado final desse trabalho, uma biblioteca de Segredo Compartilhado, de maneira livre, aberta e com as garantias necessárias para que esse não seja mais um impedimento ou um beco sem saídas a todas as comunidades que necessitem de uma ferramenta que se encaixe nesse caso.

1.4 METODOLOGIA

A busca pela completude do trabalho proposto se deu através de esforço dos envolvidos mas não apenas isso seria necessário para atingir os objetivos do mesmo.

Alguns pontos cruciais devem ser mencionados em relação ao processo de desenvolvimento, entre eles:

- Criação de um padrão na realização de testes durante o desenvolvimento
- Uso de ferramentas como *Valgrind* e *gdb* na busca de erros durante a fase de desenvolvimento e no uso da biblioteca
- Uso de scripts para testes no uso da biblioteca simulando testes de caixa preta
- Análise de desempenho com ferramentas já existentes
- Avaliação dos Resultados obtidos

- Comparação com as ferramentas já existentes

1.5 LIMITAÇÕES DO TRABALHO

O desenvolvimento da biblioteca limita-se a realizar o desenvolvimento e análise apenas do Esquema de Segredo Compartilhado proposto por Shamir. Esse processo será realizado em Software com a execução em um mesmo Hardware para a comparação com as alternativas existentes no mercado. A obtenção de resultados diferentes pode vir a acontecer no caso do uso de Hardwares aceleradores criptográficos.

Também é importante ressaltar o uso de uma biblioteca matemática de nome **NTL** para operações e manipulação dos tipos de dados da linguagem `C++` e que a otimização dessa biblioteca remete diretamente na otimização do uso da ferramenta aqui produzida. Versões diferentes da biblioteca **NTL** podem refletir em resultados diferentes ou incompatibilidade do projeto.

1.6 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado de forma a proporcionar ao leitor a aquisição dos conceitos necessários para a continuidade da leitura e compreensão do mesmo com o texto aqui apresentado.

O capítulo 2 traz conceitos básicos sobre Segurança em Computação, conceitos sobre Segredo Compartilhado e apresenta Esquemas de Segredo Compartilhado de Blakley e de Shamir contextualizando a implementação deste último por parte da biblioteca proposta.

O capítulo 3 apresenta o desenvolvimento da biblioteca, desde a estruturação de sua Arquitetura até a forma como foi testada e as escolhas de projeto tomadas. São nele apresentados trechos de código com comentários explicativos e que elucidam o leitor sobre quais ações estão sendo tomadas pela biblioteca em sua execução. É também apresentada a escolha do uso de uma biblioteca matemática chamada **NTL** para a implementação do trabalho no auxílio a funcionalidades matemáticas como por exemplo o uso de sua implementação de Corpos Finitos e suas otimizações quanto ao processamento dos dados. Tanto o código final produzido como exemplos de teste estão disponíveis na seção de anexos.

O capítulo 4 apresenta o resultado e a análise final, o processo de testes e compara a ferramenta proposta e desenvolvida com outras fer-

ramentas disponíveis em questões como limitações de uso e velocidade de processamento.

Ao final, o capítulo 5 traz as considerações finais sobre o trabalho e estabelece os pilares principais do que pode ser realizado por futuros trabalhos aproveitando o resultado do trabalho aqui proposto.

2 FUNDAMENTAÇÃO TEÓRICA

Para a análise inicial da biblioteca produzida alguns conceitos precisam ser mencionados dentro do escopo do trabalho. Este trabalho está embasado na área de estudo da Segurança em Computação, mais especificamente relacionado a Segredo Compartilhado e antes de adentrar ao escopo em si, serão apresentadas algumas definições relacionadas tanto a Segurança em Computação quanto Segredo Compartilhado.

Em primeiro momento serão apresentadas definições mais gerais Segurança em Computação para contextualizar o leitor a respeito de termos e conceitos utilizados durante o presente trabalho. Conceitos como Assinatura Digital, Privacidade, Confiabilidade e Não-Repúdio serão discutidos e elucidados com um foco mais específico no que diz respeito ao uso mais geral e o uso específico do trabalho bem como o auxílio a análise final da biblioteca. Após essa sessão inicial serão apresentados conceitos mais específicos dentro do escopo do trabalho.

Adentrando os conhecimentos de maneira mais minuciosa, será realizada uma discussão mais específica sobre Segredo Compartilhado e uma breve descrição de como são suas características. Na sequência, o conceito de Segredo Compartilhado será definido de forma matemática não deixando brechas para ambiguidades, visto que o conceito em si pode ser estendido a diversos tipos de implementações ou esquemas.

Após a definição do conceito de Segredo Compartilhado serão mostradas propostas de Esquemas como os de Blakley(BLAKLEY et al., 1979) e Shamir(SHAMIR, 1979). Enquanto Blakley propõe um Esquema baseado em vetores paralelamente ao que ele mostrou com o uso de polinômios Shamir se aprofunda no uso de polinômios e define seu Esquema levando o nome de *Esquema de Limiar*, ambos criados na busca da resolução de um problema proposto por Liu(LIU, 1968). Apesar de Blakley definir no mesmo ano e ser citado por Shamir em sua publicação, este trabalho se concentra em implementar a versão do *Esquema de Limiar* possibilitando a escalabilidade para outros esquemas serem implementados no futuro.

2.1 PRIVACIDADE

Quando iniciamos os estudos a respeito de Segurança em Computação geralmente notamos uma sobreposição de áreas dentro de Ciências da Computação. Notamos isso através do currículo de diversas

Universidades ainda usarem como referência principal o clássico texto produzido por Tanenbaum(TANENBAUM, 2009) nas disciplinas tanto de Segurança em Computação quanto em disciplinas de Sistemas Operacionais, Redes de Computadores, Sistemas Embarcados e por vezes até disciplinas de Algoritmos. Por ser um trabalho muito completo que realiza demonstrações e exemplos de técnicas e abordagens usadas através dos anos nos diversos Sistemas Operacionais conhecidos e nos Sistemas Embarcados o fato de um Sistema Operacional precisar tratar de assuntos específicos como iteração direta com o hardware, controle de memória nos diversos níveis(volátil e não-volátil) ele também traz informações importantes acerca de algoritmos de escalonamento e conceitos que podem ser aplicados em camadas mais abstratas do Sistema Operacional e que fazem muita referência a outras áreas como as já citadas.

Podemos notar então que a área de Segurança em Computação não está isolada como um iceberg que se desprendeu e está a deriva no mar. Muitos conceitos e propostas dentro da área foram feitas por profissionais de outras áreas correlatas ou de áreas que por vezes até não possuem um contato direto com a área da Segurança.

É notável também que outros autores fazem o uso de citações que abrangem mais de uma área em seus trabalhos na área da Segurança. É o caso de muitos conceitos que serão propostos para elucidar o leitor em sua trajetória pelo texto.

2.2 CONCEITOS BÁSICOS

Como literatura recomendada nas disciplinas de Segurança em Computação, Stallings(STALLINGS, 2005) nos traz alguns conceitos muito válidos quando cita Shirley(SHIREY, 2000) em relação aos conceitos da **Arquitetura de Segurança OSI**. Essa Arquitetura foi desenvolvida como um padrão internacional e que tem como fim a interoperabilidade entre os produtos de comunicação e rede de diversos fornecedores diferentes no que diz respeito a comunicação e troca de informações e mensagens.

Stallings comenta o fato de que essa arquitetura traz diversos conceitos válidos para seu texto bem como são ao leitor do trabalho em questão.

- **Ataque de Segurança**: Qualquer ação que compromete a segurança da informação que uma organização detém. O **Ataque de Segurança** é a concretização de uma **Ameaça de Segurança**

- **Mecanismo de Segurança**: Um processo ou um dispositivo que incorpora o processo desenvolvido para detectar, prevenir ou recuperar de um **Ataque de Segurança**

- **Serviço de Segurança**: Um serviço de processamento ou comunicação que aumenta a segurança de um sistema de processamento de dados. Geralmente fazem uso de um ou mais de um **mecanismos** para proporcionar segurança

Enquanto os **Serviços de Segurança** tratam de questões como Autenticação, Controle de Acesso, Integridade e Confidencialidade dos Dados e o Não-Repúdio eles fazem uso de **Mecanismos de Segurança**: Assinatura Digital, Recuperação de Segurança e Cifragem. Também para o auxílio ao leitor durante o decorrer do trabalho é válido comentar alguns conceitos referentes a Mecanismos de Segurança visto que o produto final do mesmo vem a ser uma implementação de um desses Mecanismos para garantia da Segurança de um Sistema.

A proposta de um Esquema de Segredo Compartilhado vem como um Mecanismo de Segurança de Cifra e de Controle de Acesso a um *recurso*. Isso significa que em primeiro momento deve existir um *recurso* em forma de documento digital e a existência de uma motivação de controle de acesso a esse recurso. Esse recurso pode ser um documento em texto plano, uma imagem até um documento sigiloso que possui informações confidenciais como uma Chave Criptográfica, por exemplo. Quando existe essa motivação com relação ao controle de acesso desse recurso existem abordagem que podem ser tomadas com o uso de diversos Mecanismos de maneira singular ou composta.

Como exemplo podemos citar um caso onde existe uma troca de mensagens eletrônicas via Internet de duas partes. Por questão de comodidade vamos considerar as duas partes duas pessoas: Ana e Bob. A forma mais simples de realizar a troca de mensagens seria apenas o envio de uma mensagem eletrônica diretamente aos seus respectivos correios eletrônicos, de Ana para Bob e o inverso.

Essa simples troca de mensagens poderia ser interceptada por um atacante, lida e eventualmente até modificada pelo mesmo caso não seja realizado o uso de um ou mais Mecanismos de Segurança. Como primeira forma de Mecanismo de Segurança podemos citar o uso do OpenPGP, definido em (CALLAS et al., 2007) e (CALLAS et al., 1998) que se trata de uma melhoria feita a partir do PGP(ATKINS; STALLINGS; ZIMMERMANN, 1996), um Formato de Troca de Mensanges entre partes visando a garantia de **Privacidade** e **Autenticação** em trocas de mensagens. De maneira bem abstrata, o OpenPGP realiza

suas premissas através de uma sequência de operações com base na mensagem que está sendo enviada e uso de Assinatura Digital.

Como primeira forma de autenticação o OpenPGP agrega muito valor a alternativa de apenas enviar o texto em claro e ataques começam a ser mitigados com seu uso e a tarefa inicial de dar mais segurança para a troca de mensagens entre Ana e Bob pode ser atendida com maior completude. Por outro lado o OpenPGP não está livre de ataques e um ataque simples seria apenas conseguir a Chave de Ana e enviar mensagens a Bob com ela e perante toda a infraestrutura criada pelo OpenPGP a mensagem ainda assim seria válida. Outras alternativas poderiam ser aplicadas para garantir maior segurança porém a discussão dessas alternativas não vem a ser o propósito desse trabalho.

2.3 CRIPTOGRAFIA

Se torna importante também uma discussão acerca de Algoritmos de Cifragem em sua maneira mais abstrata tanto para Cifras Simétricas quanto Assimétricas. Alguns conceitos aqui citados também possuem direta correlação com o trabalho e uma apresentação desses conceitos se faz válida visando a compreensão subsequente do texto.

Em primeiro momento serão apresentados conceitos essenciais para a compreensão básica de um Esquema de Cifragem. Serão apresentados esses conceitos essenciais de maneira geral ainda que eventualmente alguns deles não façam parte de ambos os Esquemas descritos na sequência. São eles

- **Texto Plano**: Também chamado de *Texto em Claro*, é a mensagem original ou o dado que é inserido como entrada no Algoritmo
- **Algoritmo Criptográfico**: Também é chamado de Algoritmo de Cifragem, realiza diversas operações sobre o Texto Plano sejam elas operações lógicas simples ou compostas com algum tipo de *Semente* ou Chave Criptográfica. Pode ser classificado como
  - *Algoritmo Criptográfico Simétrico* é o caso em que apenas uma Chave é necessária para realizar a Cifra e para Decifrar
  - *Algoritmo Criptográfico Assimétrico* é o caso onde existe um Par de Chaves e elas devem ser utilizadas de maneira alternada: enquanto uma delas Cifra a outra Decifra
- **Chave Criptográfica**: Serve também como entrada no Algoritmo de Cifragem e é um valor independente tanto do Algoritmo

quanto do Texto Plano. O uso de duas Chaves diferentes sobre o mesmo Texto Plano e o mesmo Algoritmo produz duas saídas totalmente diferentes bem como o uso de uma mesma Chave para dois dados diferentes usados como Texto Plano

- **Par de Chaves**: Para o caso do uso de um *Algoritmo Criptográfico Assimétrico* são geradas duas chaves diferentes mas que se completam no que diz respeito a produção do Texto Cifrado a partir do Texto Plano e vice-versa

- **Texto Cifrado**: É a saída do Algoritmo de Cifragem e depende diretamente do Texto Plano e da Chave Criptográfica. Ainda que dependa diretamente do Texto Plano, é ininteligível e aparentemente parece como uma sequência aleatória de dados

- **Algoritmo de Decifragem**: É o algoritmo que toma como entrada uma Chave Criptográfica e o Texto Cifrado e produz como saída o Texto Plano. Caso o Algoritmo Criptográfico seja *Simétrico* a Chave Criptográfica deve ser a mesma que produziu o Texto Cifrado. No caso de um Algoritmo Criptográfico Assimétrico a Chave Criptográfica que deve ser utilizada deve ser a outra Chave Criptográfica referente ao Par

É importante citar também que um Sistema Criptográfico é caracterizado por três dimensões independentes e que garantem seu funcionamento. O **tipo de operações** utilizadas para transformar o Texto Plano em Texto Cifrado podem fazer referência a operações de substituição ou de transposição dos elementos do Texto Plano, sejam eles bits, bytes ou grupos maiores de agrupamento. O **número de chaves** utilizadas no sistema também influencia diretamente na forma que as operações ocorrem sendo dito que com o uso de apenas uma Chave Criptográfica a chave é chamada de Chave Simétrica e no caso de um Par de Chaves elas são chamadas de Par de Chaves ou Chaves Assimétricas. Já a **forma que o Texto Plano é processado** diz respeito a como são organizados os dados processados no decorrer da operação e pode ocorrer através de cifradores de *blocos* ou com os cifradores de *stream*.

2.4 ALGORITMOS CRIPTOGRÁFICOS SIMÉTRICOS

Munido dos conhecimentos até agora apresentados é possível discutir alguns Algoritmos que trazem efetividade a Serviços de Segurança.

Apresentados por Stallings e baseados em **Substituição**, **Transposição** ou trabalhar com auxílio de **Rotores**, esses Algoritmos trouxeram conceitos importantes através dos anos sendo importantes em diversos momentos históricos conforme novas tecnologias eram desenvolvidas. Hoje ainda são largamente utilizados e possuem papel importante nos Serviços de Segurança oferecidos pelas mais gerais Aplicações existentes.

Como uma das mais simples técnicas existentes, **Algoritmos de Substituição** possuem como sua principal característica a simples substituição de caracteres. Eles foram talvez uma das primeiras formas de comunicação segura criada e o seu funcionamento é simples e o conhecimento de um de seus primeiros mecanismos levou o nome de seu criador histórico: a **Cifra de César**. Ele caracterizado por ser um Cifrador **Monoalfabético** e é utilizado como exemplo como uma forma simples de comunicação segura. A **Cifra de Playfair** é um outro algoritmo **Monoalfabético** muito conhecido.

Como exmpelo de uma técnica de cifra utilizando mais de um alfabético podemos citar o Cifrador **Poli-alfabético** chamado de **Cifra de Vigenère** que utiliza uma tabela para realizar a Cifra do *Texto em Claro*.

Por mais que alguns considerem essa técnica simples, hoje ainda existem estudos sendo realizados com essa técnica e que demonstram a constante evolução e o reaproveitamento de conhecimentos por vezes considerados ultrapassados para a criação e evolução da fronteira do conhecimento de hoje. Como exemplo desse constante estudo podemos citar o estudo de Abraham(ABRAHAM; SHEFIU, 2012) com o desenvolvimento do *ICC - Improved Caesar Cihper* que elimina o uso de espaços no Texto Cifrado e melhora a forma de escolha de letras no alfabeto secundário. Também podemos citar outro estudo recente de Goyal(GOYAL; KINGER, 2013) que faz uso de um cálculo do valor do índice da letra no alfabeto original e realiza um acréscimo ou decréscimo na distância do *shift* executado, aumentando assim a segurança do algoritmo original.

Outra técnica que podemos citar como notória na evolução dos Algoritmos Simétricos são os *Algoritmos de Transposição*. As Técnicas de Transposição caraterizam-se pela transposição da mensagem dentro de suas próprias letras e não substituindo as mesmas por outras baseadas em um alfabeto como a técnica de substituição.

Ainda com relação aos Algoritmos Simétricos podemos citar os Cifradores de Blocos, que utilizam uma parte fixa de um texto chamada de "bloco"e realizam operações sobre esses "blocos"uma sequencia de

operações lógicas. Até hoje já foram estudados e propostos diversos **Cifradores de Blocos** como o **DES - Data Encryption Standard** e o **AES - Advanced Encryption Standard**.

É importante citar também que ao mesmo tempo que novos Algoritmos são construídos e criados existe um campo em constante estudo chamado de **Criptoanálise** que estuda possíveis ataques e falhas nos Algoritmos até agora criados. Como exemplo, é possível citar um estudo realizado por Biham e Shamir(BIHAM; SHAMIR, 1991) onde é comentado sobre um ataque de **Criptoanálise Diferencial**, uma subcategoria da **Criptoanálise** que se caracteriza pela observação de pares de blocos avançando a cada round de informação sendo Cifrada por um Cifrador de Blocos como o DES.

2.5 ALGORITMOS CRIPTOGRÁFICOS ASSIMÉTRICOS

De forma similar aos **Algoritmos Criptográficos Simétricos** onde o mecanismo funcionava aceitando apenas uma mesma **Chave Criptográfica** para ambos os processos de Cifra e de Recuperação do Segredo (Decifrar), os **Algoritmos Criptográficos Assimétricos** possuem como característica o fato de que dependem de um **Par de Chaves**. Essas Chaves são um par pois elas funcionam de forma a que uma operação realizada com uma das Chaves é desfeita com o uso da Chave par equivalente.

Stallings também classifica esses Algoritmos como a categoria de **Algoritmos de Chaves Públicas** derivado do uso das Chaves. Após a geração das mesmas, uma delas pode ser distribuída para aquele que desejam enviar informações sensíveis ao criador do Par de Chaves e leva o nome de **Chave Pública**. A outra Chave do par, também chamada de **Chave Privada**, é mantida em segredo e será usada para o envio de informações sensíveis para os detentores da *Chave Pública*.

É importante mencionar que apesar do Par de Chaves ter sido gerado no mesmo momento ele não possui qualquer correlação que possibilite a descoberta de uma das Chaves apenas com o conhecimento da outra do mesmo Par. Dessa forma é possível criar um canal seguro de comunicação entre duas partes e que garanta graus maiores de confiabilidade em relação ao uso de um Algoritmo Simétrico.

Um dos Algoritmos de Criptografia Assimétricos mais conhecido é o algoritmo proposto por Rivest, Shamir e Adleman (RIVEST; SHAMIR; ADLEMAN, 1978) e que mais tarde levaria o nome de seus criadores: **RSA**. O **RSA** também foi proposto a comunidade como o primeiro

Public-Key Criptographic Standard, ou apenas **PKCS** #1 em forma de RFC (KALISKI, 1998), e em sua publicação de versão mais recente (MORIARTY et al., 2016).

A ideia principal que fundamente o uso de um **Algoritmo de Criptografia Assimétrico** é o fato de que em vez de se ter uma única Chave para a comunicação entre duas entidades e que com a eventual descoberta dessa Chave torne necessária a realização de um novo procedimento de Troca de Chaves existam duas Chaves: Uma **Pública** que deve ser aberta a todos aqueles que querem se comunicar de maneira segura com a entidade que distribui a chave e outra **Privada** que deverá ser guardada em segredo e que possibilita a todos que recebem uma informação sensível tenham a completa certeza de que a entidade que criou aquela informação é inegavelmente a detentora da Chave **Privada** correspondente a Chave **Pública** que realiza a operação de descoberta dessa informação sensível.

Com um bom Mecanismo Criptográfico e utilizando Criptografia de Chaves Públicas é possível realizar a garantia de **Autenticidade** e **Não-repúdio**. É comum o uso do RSA em Mecanismos de **Assinatura Digital** com o auxílio de funções de **Resumo Criptográfico** (Hash) para que essas garantias sejam viabilizadas.

2.6 ASSINATURA DIGITAL

A necessidade de um Mecanismo de Autenticação robusto e que desse suporte a integridade da mensagem foi talvez um dos mais importantes gatilhos que impulsionou o estudo e a proposta da Assinatura Digital e até hoje torna essa área um vasto campo de estudo dentro da Segurança em Computação. Como importante revisão bibliográfica acerca de Assinatura Digital é possível citar o estudo de Aki (AKI, 1983) que foi usado como uma referência também por Stallings em sua literatura.

2.7 ASSINATURA DIGITAL E PROTOCOLOS DE AUTENTICAÇÃO

Quando é falado especificamente sobre **Assinatura Digital** estamos citando um mecanismo de segurança que possui algumas propriedades e alguns requerimentos para seu funcionamento e algumas abordagens e protocolos para a garantia de autenticação entre as par-

tes envolvidas.

No que diz respeito das propriedades de **Assinatura Digital** Stallings enumera

- A assinatura deve verificar o autor da mensagem, a data e o momento que foi assinado
- A assinatura deve autenticar o conteúdo no momento em que foi assinado
- A assinatura deve ser verificada por terceiros, visando a resolução de disputas

Quanto aos requerimentos para uma **Assinatura Digital**, baseado nas propriedades acima mencionadas temos

- A assinatura deve possuir um padrão e depender diretamente da mensagem assinada
- A assinatura deve usar alguma informação única de quem envia, prevenindo mensagens forjadas e o não-repúdio
- Deve ser relativamente fácil reconhecer e verificar a assinatura
- Deve ser computacionalmente inviável de forjar a **assinatura digital**, tanto por construir uma nova mensagem para uma assinatura existente ou construir uma assinatura digital fraudulenta para uma mensagem enviada
- Deve ser simples o armazenamento de uma cópia da **assinatura digital**

Utilizando essas propriedades e esses requerimentos é possível estabelecer algumas abordagens para a forma que o protocolo deva funcionar. Entre as abordagens propostas, Stallings cita o fato de que elas sempre residem em duas categorias:

- **Assinatura Digital Direta**
- **Assinatura Digital Arbitrada**.

A categoria de **Assinatura Digital Direta** envolve apenas as partes que estão se comunicando(origem e destino) e é assumido que a parte de destino conhece a *Chave Pública* da parte de origem. A mensagem assinada é construída cifrando o a própria mensagem original

em sua totalidade ou cifrando o Hash da mensagem original com a *Chave Privada* de quem está enviando a mensagem.

A Confidencialidade pode ser obtida através da cifra da mensagem em sua totalidade e da assinatura usando a *Chave Pública* do destino ou usando uma *Chave Simétrica* compartilhada.

É importante citar ainda sobre a **Assinatura Digital Direta** o fato de que a validade do Esquema reside diretamente na segurança da *Chave Privada* da parte que envia a mensagem(origem). Eventualmente a entidade que realiza o envio da mensagem pode também declarar que sua *Chave Privada* foi perdida ou roubada e que a mensagem enviada é falsa.

Quando observamos a categoria de **Assinatura Digital Arbitrada** conseguimos notar que com o seu funcionamento usando um árbitro resolve os problemas da categoria anterior. A ideia principal dessa categoria é o envio prévio da mensagem que será enviada de uma parte de origem a uma parte de destino para um árbitro e ele realiza uma série de testes para verificar a origem da mensagem e do seu conteúdo para só então ser enviada ao destino com uma informação de que ela foi verificada pelo árbitro.

Nessa categoria deve existir um grande grau de confiança entre as duas partes e a entidade do árbitro para que elas confiem em sua presença e decisões.

Avançando para o campo de **Procolos de Autenticação** é possível a discussão sobre duas áreas mais gerais e que possuem um grande impacto dentro da área de Assinatura Digital: **Autenticação Mútua** e **Autenticação de Via-Única**.

Na **Autenticação Mútua** é usada uma *Chave de Sessão* para a conversa entre as duas partes envolvidas. Para garantia de **confidencialidade** a autenticação acontece com uma troca de chaves para que a *Chave de Sessão* seja estabelecida e essa troca ocorre através de um canal seguro, podendo este ser a existência de um par de chaves. Já para a garantia de que mensagens não sejam repetidas e dê oportunidade a atacantes de interceptar e encontrar padrões, é atrelada a mensagem uma sequencia numérica que pode representar por exemplo o momento em que a mensagem foi enviada.

A abordagem de **Autenticação Mútua** pode utilizar de Criptografia Simétrica ou Assimétrica, podendo também fazer uso de um Árbitro ou não.

Com relação a **Autenticação de Via-Única**, é possível citar que ela não exige que as duas partes estejam online ao mesmo tempo para que ocorra. Um bom exemplo disso é o uso de autenticação via

mensagens eletrônicas (e-mails). Ela também pode fazer uso de Criptografia Simétrica ou Assimétrica mas o uso de um Árbitro se torna inviável visto que as duas partes podem não estar prontas para realizar a requisição pela *Chave de Sessão* ao mesmo tempo, podendo ainda demorar um tempo indefinido para ocorrer a abertura da mensagem e sua verificação pela entidade que recebe a mesma.

É importante citar ainda que foram criados durante o passar dos anos alguns padrões de Assinatura Digital, com seu resultado mais expressivo o Padrão criado pelo *National Institute of Standards and Technology (NIST)* em sua publicação (MEHURON, 1994) estabelecendo também um **Algoritmo de Assunatura Digital - DSA** e que mais tarde foi expandido para o uso de **RSA** e **Curvas Elípticas** (FIPS, 2000) e (FIPS, 2009).

2.8 GERENCIAMENTO DE CHAVES

Sempre motivadas pela manutenção da confidencialidade na troca de mensagens entre duas ou mais partes os estudiosos estabeleceram padrões robustos e com garantias importantes dentro do campo da segurança como não-repúdio e confiabilidade. Pela ótica de uma entidade que faz uso desses padrões ela se protege muito bem de diversas falhas e ataques conhecidos e pode continuar aprimorando seus processos para que sua segurança seja ainda maior a cada mudança ou resoluções de novas falhas.

Entretanto os avanços na área da Assinatura Digital dependem essencialmente de um armazenamento seguro do material mais sensível aos olhos de todo o Mecanismo: **Chave Privada** de uma entidade. Não importa o quão seguro o Mecanismo for nem quantas camadas de segurança forem colocadas se a **Chave Privada** está salva em um hardware simples e de fácil acesso como o Disco Rígido de um computador ligado a internet e sem nenhuma camada de segurança.

Ainda de maneira mais crítica, podemos mencionar o caso de que existem entidades que podem ser representadas por um número de indivíduos maior do que um podendo ser um conjunto de Nodos em uma Rede de Sensores Sem-Fio ou um conjunto de Sócios de uma Empresa. Essa situação apenas piora quando mencionamos o fato de que eventualmente um desses indivíduos pode ser comprometido de maneira intencional ou não por algum atacante e abordagens como a manutenção de uma **Chave Privada** por todos os membros da entidade pode se tornar crítica.

Visando esse estudo e esse foco na proteção de um **material sensível** tal como uma **Chave Privada** já é presente na literatura uma solução que será apresentada na sequência em forma teórica e implementada como uma solução viável para uso de uma entidade que tenha como necessidade esse caso de uso.

2.9 SEGREDO COMPARTILHADO

Existem diversos problemas cotidianos onde reside a necessidade do uso de uma mesma *Chave Criptográfica* por mais de uma entidade sejam elas pessoas, máquinas ou programas.

- Sócios de uma empresa realizando a assinatura de um relatório confirmando seu aceite para determinado processo
- Chefes de nações confirmando um inicio de ataque a algum inimigo em potencial
- Donos de uma mesma conta bancária confirmando uma transação de um grande montante

Nestes casos a geração e uso de várias Chaves Criptográficas (uma para cada entidade envolvida) se torna um tanto quanto inviável. Pode-se ter uma melhor percepção deste custo quando enumeramos os problemas que podem vir a ocorrer em virtude do uso deste sistema.

- Custo de processamento na validação constante durante operações realizadas
- No caso eventual de invalidação de uma das chaves (seja por perda ou vazamento de informações) existe a necessidade da realização de uma cerimônia para a nova distribuição desta chave
- A ordem de entrada das chaves no momento de abertura do segredo deve ser **exatamente** a ordem inversa da ordem de entrada no momento que o dado foi cifrado visto que

$$dado = Cifrar(Decifrar(dado)) = Decifrar(Cifrar(dado))$$

No caso específico do uso de apenas uma Chave Criptográfica por mais de uma entidade dentro de um Sistema Criptográfico algumas abordagens são tomadas para que uma eventual perda ou comprometimento de parte do sistema possa ser replicado. Essas abordagem geralmente residem em manutenção de cópias do material Criptográfico

em armazenamentos *não-voláteis* e mantidos por entidades de confiança dos envolvidos no Sistema visto que alguma falha pode ocorrer e o Sistema todo pode ser comprometido com a perda de uma Chave Criptográfica.

Essa tomada de decisão acarreta uma série de outras possíveis falhas e oportunidades a atacantes que favorecem eventuais quebras de segurança bem como algumas falhas relacionadas a manutenção desse armazenamento não-volátil. Essas falhas envolvem o comprometimento intencional ou não desse material criptográfico e quando generalizados é possível notar um padrão e subdividir essas falhas em 3 tipos de incidentes como Blakley(BLAKLEY et al., 1979) discute:

- *Incidente de Abnegação*: Um evento em que a informação ou o dado *não-volátil* não está mais disponível para ser recuperada em sua totalidade. Dentro dessa categoria existem também três sub-categorias em que um incidente de Abnegação se enquadra:
  - *Destruição*: A informação ou dado é destruída em parte ou em sua totalidade, sendo este intencional ou não
  - *Degradação*: A informação ou dado é perdida e uma outra informação precisa ser gerada para que a entidade responsável obtenha a sua parte não-volátil e o sistema continue com o mesmo grau de replicação e segurança
  - *Deserção*: A informação ou o dado é divulgado pela entidade e essa mesma entidade não comunica aos outros membros envolvidos no sistema a respeito desse comportamento
- *Incidente de Traição*: Um evento em que a informação ou o dado *não-volátil* é informado em sua completude a um atacante, um opositor a entidade que possui o segredo ou a entidades que se beneficiariam de maneira negativa. Vale ressaltar que um incidente de *Deserção* é também um tipo de Incidente de Traição. Uma outra sub-categoria em que um Incidente de Traição se enquadra também é:
  - *Desamparo*: A informação ou o dado é divulgado pela entidade responsável a um oponente e essa mesma entidade continua atuando dentro do sistema normalmente e reportando a sua informação ou dado de maneira que não levante suspeitas. Aqui o Desamparo faz relação a maneira como a entidade se comporta perante o seu papel: Desamparo pelo papel de importância dentro do sistema.

- *Incidente de Combinação*: Um evento combinado em que o mesmo é um Incidente de Abnegação e de Traição. Um exemplo é o *Incidente de Deserção.*

É importante ressaltar também que nenhum dos tipos de Incidentes implicam diretamente em vontade intencional da entidade envolvida de realizar um ato que tenha como fim prejudicar o Sistema como um todo. Todavia, sempre se deve considerar esse tipo de ocorrência quando se analisa as questões envolvidas que garantem segurança e confiabilidade do Sistema.

2.9.1 Definição

No problema proposto por Liu(LIU, 1968): *Onze cientistas estão trabalhando em um projeto secreto. Eles querem guardar os documentos da pesquisa em um armário com fechaduras para que o armário só possa ser aberto caso seis ou mais cientistas estão presentes. Qual é o menor número de fechaduras necessárias? Qual é o menor número de chaves para as fechaduras que cada cientista precisa ter consigo?*

Blakley(BLAKLEY et al., 1979) discute sobre esse problema e Shamir(SHAMIR, 1979) nos apresenta que a solução: São necessárias **252 chaves** que cada cientista deveria guardar consigo e a existência das **462 fechaduras**. É perceptível que esta solução não está nem um pouco perto da solução ideal e o crescimento dela é exponencial para casos maiores segundo Shamir.

2.9.2 Esquema de Blakley

Baseado nos tipos de Incidentes comentados previamente, Blakley(BLAKLEY et al., 1979) coloca em seu trabalho uma análise com base nos Incidentes tomando como relação:

- a = número de Incidentes de Abnegação
- b = número de Incidentes de Traição
- c = número de Incidentes de Combinação

Assim, teríamos como definir o *Primeiro Princípio na Contagem de Incidentes.* Utilizando a *Lei de Boole* de inclusão e exclusão, teríamos o total de incidentes d sendo obtido através do cálculo

$$d = a + b - c$$

Onde o valor de Incidentes de Combinação c é subtraído visto que eles estão sendo contados duplicados em a e em b. Também em seu Esquema, os valores a, b, c e $a + b - c$ não devem exceder o número de cópias *não-voláteis* da informação, denotado por g.

O *Segundo Princípio na Contagem de Incidentes* deriva do fato de que incidentes são tão raros que dois incidentes separados acontecendo com a mesma parte de informação não-volátil é geralmente deixada de lado visto que as probabilidades de ocorrência beiram quase o absurdo. Dessa forma, é possível dizer que o tamanho de c geralmente é desconsiderado ou pode ser calculado com a incidência mínima de a e b ou de maneira matemática

$$c \leq \min\{a, b\}$$

Assumindo então que uma organização queira realizar uma cerimônia de divisão de uma informação sigilosa em **partes**, a mesma deve considerar um número de a incidentes de abnegação e um número de b incidentes de traição de que se deve estar protegido contra no processo de recuperação dessa informação sigilosa envolvendo um número g de entidades atuando como *guardas* dessas **partes**. As **partes** de uma informação possuem mais algumas propriedades que serão discutidas mais a frente mas por hora, serão consideradas as *partes* da informação que auxiliam em sua reconstrução. O tempo de vida desse esquema também não deve ser algo muito além de alguns meses se incidentes separados envolvendo uma mesma **sombra** serão descartados.

A partir dos dois princípios descritos acima pode-se deduzir que

$$a + b - \min\{a, b\} \leq d \leq a + b$$

E tendo $d \leq g$, onde g é o número de *guardas* que tiveram as g **partes** da informação.

Também é comentado por Blakley que em virtude da prudência e da previsão dos incidentes, é possível desconsiderar o valor c, onde teríamos

$$d = a + b - c = a + b + 0$$

Com essa abordagem e considerando que teríamos a informação

como uma *Chave Criptográfica*, é necessária a geração de pelo menos $a+b+1$ **partes** que realizariam a reconstrução da chave e essas partes devem ser distribuídas entre *guardas* diferentes. Assumindo que a são os incidentes de abnegação e b são os incidentes de traição, essa reconstrução deve acontecer apenas com o mínimo de $b+1$ **partes** e com uma quantidade igual ou menor a b não deve ser possível a recuperação ou a inferência de nenhuma informação acerca da Chave.

Blakley relata que essa último requerimento específico de que $b+1$ partes da informação deve ser capazes de reconstruir o a informação original mas que com apenas b partes nenhuma informação deve ser possível de recuperação não é usual. Caso se queria utilizar polinômios, são conhecidos métodos de recuperação de um polinômio de grau b com no mínimo $b+1$ elementos entretanto, com apenas b elementos ainda é possível a obtenção de informações acerca do polinômio original.

A frente, Blakley descreve uma proposta para a implementação desse esquema fazendo alusão às partes necessárias para a reconstrução da informação chamando as mesmas de **sombras** visto que as mesmas não são apenas partes exatas da informação quebradas e não possibilitam qualquer conhecimento sobre a informação original. Blakley usa como exemplo de informação sigilosa uma *Chave Criptográfica* e também mantém as premissas de que existem $a+b+1$ **sombras** da chave e que com $b+1$ **sombras** é possível realizar a reconstrução. Para tal, ele descreve a correspondência entre Chaves Criptográficas e um subespaço vetorial unidimensional de um espaço vetorial finito F. Uma Chave Criptográfica pode determinar uma coleção grande de vetores unidimensionais (linhas) mas um vetor unidimensional (uma linha) determina uma pequena quantidade de Chaves.

O procedimento de distribuição de **sombras** de uma Organização que possui uma Chave em uma quantidade de $a+b+1$ guardas acontece de forma que primeiro se escolhe uma das linhas correspondente a sua Chave, chamada de L. Então se escolhe um subespaço vetorial aleatório entre os $a+b+1$ existentes de f, que são as **sombras** da chave de forma que uma quantidade b ou menor dessas **sobras** intersecte uma grande quantidade de subespaços vetoriais do espaço vetorial original F e também que essa quantidade b ou menor de subespaços faça correspondência com todas as possíveis Chaves com igual probabilidade mas que $b+1$ desses subespaços intersecte a linha L. Uma vez que a linha L foi encontrada existem apenas algumas possibilidades de Chaves que podem ser a Chave original e todas essas Chaves encontradas, que agora são uma quantidade pequena, são testadas em uma lista de blocos de texto em claro até que a Chave original seja

identificada.

Em seu Esquema, Blakley faz uso de Espaços e Subespaços Vetoriais para garantir as propriedades requeridas para um bom sistema de compartilhamento de um segredo por uma quantidade finita de entidades após relatar o fato de que o uso apenas de Polinômios não traz consigo todas as garantias necessárias para esse sistema funcionar de maneira ideal. Veremos a seguir o Esquema proposto por Shamir e que faz o uso de Polinômios com melhorias. É também esse Esquema relatado a seguir o que foi priorizado na implementação do trabalho em questão.

2.9.3 Esquema de Shamir

Shamir relata em seu trabalho a proposta de Blakley mas escolhe seguir por outro caminho. Em sua abordagem, em vez de se basear em possibilidades de Incidentes e a utilização de Espaços e Subespaços Vetoriais para fazer a divisão e recuperação da informação ele opta por utilizar Polinômios com algumas melhorias em relação a sua aritmética. Assim, as propriedades do Esquema são garantidas e uma outra forma de implementação é proposta.

Mais especificamente buscando a solução ao problema de Liu, Shamir primeiro generaliza o problema e nos apresenta o que ele chamou de **Esquema de Limiar** como: (k, n) *threshold scheme*.

Definição 2.9.1. Esquema de Limiar: Queremos compartilhar um dado D em n partes $D_1, D_2, \ldots, D_n$ de forma que as seguintes propriedades sejam respeitadas:

i) com o conhecimento de k ou mais partes D_i é facilmente recuperável o dado original D

ii) com o conhecimento de $k-1$ ou menos partes D_i não é possível recuperar qualquer informação sobre o dado original D

Neste momento podemos observar o seguinte caso tomando de maneira genérica $n = 10$ e $k = 6$, uma Organização que quer distribuir uma Chave Criptográfica D entre seus 10 membros do conselho de tal forma que um número de membros menor que 6 não possa obter nenhuma informação sobre a Chave D e com exatos 6 ou mais membros presentes a Chave D possa ser recuperada. Essa Organização pode fazer uso do Esquema de Limiar.

Quando observado mais a fundo, o Esquema pode trazer consigo as propriedades necessárias para o mesmo fim que o Esquema proposto

por Blakley. Para tal, o uso do **Esquema de Limiar** com o valor $k > 0$ e respeitando a condição $n = 2k - 1$ conseguimos um esquema robusto com as seguintes propriedades:

1) com a eventual destruição ou perda de $\lfloor \frac{n}{2} \rfloor = k - 1$ partes de n, o segredo D ainda pode ser recuperado

2) com o eventual comprometimento de $\lfloor \frac{n}{2} \rfloor = k - 1$ partes de n, um atacante não consegue recuperar nenhuma informação sobre o segredo D

Para que essas propriedades sejam atendidas é proposto então que usemos um Polinômio para a divisão do segredo D. Sabemos que a construção de um polinômio se dá através de um conjunto de valores a_i tido como coeficientes e fixos e de outro valor que será avaliado dentro do polinômio para cada entrada x e que será elevado a potências i para cada valor dentro do grau do polinômio. É dito que para cada entrada x dentro de um universo, o polinômio gera gera uma saída única y. Geralmente o polinômio é denotado por uma letra do alfabeto, por exemplo q, onde podemos dizer que a avaliação de x leva ao valor y tal que

$$q(x) = y$$

Assumindo o universo dos números naturais $\mathbb{N}$ e tendo $i, x, a_i \in \mathbb{N}$ é possível denotar um polinômio de grau $i = 2$ da seguinte forma

$$q(x) = a_0 * x^0 + a_1 * x^1 + a_2 * x^2$$

Tendo como valores a_i sendo $a_0 = 2, a_1 = 3, a_2 = 5$ e omitindo as operações de multiplicação teríamos a expressão matemática

$$q(x) = 2 + 3x^1 + 5x^2$$

Caso uma avaliação de um dos valores x aconteça, por exemplo $x = 3$ é simples de se observar o resultado

$$q(3) = 2 + 3 * (3)^1 + 5 * (3)^2$$

$$q(3) = 2 + 3 * (3) + 5 * (9)$$

$$q(3) = 2 + 9 + 45$$

$$q(3) = 56$$

Pode-se afirmar então que $q(3) = 56$ para o polinômio $q(x)$, e essa avaliação nos oferece como resultado uma tupla (x, y) onde sempre que avaliarmos o valor $x = 3$ dentro do polinômio $q(x)$ teremos o valor $y = 56$ e a tupla será $(3, 56)$.

Alguns fatos importantes dessa discussão são:

- Existem infinitos polinômios $q_j(x)$ que podem nos remeter a tupla $(3, 56)$
- Apenas com o conhecimento da tupla $(3, 56)$ pouco ou quase nada se sabe sobre o grau i do polinômio ou mesmo sobre quais são os valores a_i do polinômio

É então possível observar uma direta correlação com os requisitos para a resolução do problema de compartilhamento de um dado sigiloso. Entretanto é necessário antes discutir uma forma de recuperar o polinômio $q'(x)$ a partir de um *conjunto de tuplas* (x, y) por ele gerada. Entre as formas que existem, uma se destaca tendo uma propriedade específica com relação a um número mínimo de tuplas necessárias para o polinômio ser recuperado: **Interpolação Polinomial**. Essas tuplas podem também ser denotadas como *coordenadas no espaço* visto que em uma representação gráfica um polinômio com apenas uma variável x possível de avaliação produz uma função em um Plano Cartesiano de duas Coordenadas.

Tendo agora este relacionamento de forma aproximada através de propriedade similares, podemos então definir a técnica de recuperação do Polinômio $q(x)$ baseado nas tuplas (x, y) por ele geradas

Definição 2.9.2. Interpolação Polinomial: Dado k pontos em um plano bidimensional

$$(x_1, y_1), (x_2, y_2), \ \ldots, (x_k, y_k)$$

com valores x distintos, existe **um e apenas um** polinômio $q(x)$ de grau $k - 1$ tal que

$$q(x_i) = y_i$$

para todo i.

Tendo conhecimento de um método de recuperação de um polinômio $q(x)$ com base em suas Coordenadas e assumindo que o polinômio $q(x)$ possui grau $i = k - 1$, é importante ressaltar algumas outras propriedades notáveis do uso de polinômios:

- Existe um conjunto $\mathbb{A}$ infinito de Coordenadas (x, y)
- Utilizando um subconjunto qualquer finito $A \subset \mathbb{A}$ de tamanho i é possível recuperar o polinômio $q(x)$
- Utilizando um subconjunto qualquer finito $A' \subset \mathbb{A}$ de tamanho menor que i não se faz possível a recuperação do polinômio $q(x)$

Essas propriedades salientam ainda mais a possibilidade e viabilidade do uso de Polinômios como mecanismo do Esquema de Limiar. Após essa breve discussão e agora munido com esses conhecimentos é possível dar continuidade ao que Shamir propos em seu Esquema.

Sem perda de generalidade, se pode assumir que o segredo D é, ou pode ser transformado em um número. Para dividir D em suas partes D_i, se escolhe um polinômio aleatório de grau $k-1$

$$q(x_i) = a_0 + a_1 x + \ \ldots \ + a_{k-1} x^{k-1}$$

onde $a_0 = D$ e calculamos os valores D_i com $i \in \{1, n\}$ da seguinte forma

$$D_1 = q(1), \ \ldots, D_i = q(i), \ \ldots, D_n = q(n)$$

Assim, assumindo que $\mathbb{A}$ é o conjunto infinito com todas os pontos (x, y), dado qualquer subconjunto $A \subset \mathbb{A}$ de tamanho k de D_i valores com seus pontos $(x_i, y_i) \in A$ torna possível a recuperação do polinômio $q(x)$ por interpolação para então recuperar $D = q(0)$. Como discutido previamente o conhecimento de qualquer subconjunto $A' \subset \mathbb{A}$ de tamanho $k-1$ ou menor dos valores de D_i não torna possível a recuperação de $q(x)$ e por consequência não é possível recuperar $D = q(0)$.

Para tornar essas propriedades mais precisas, é utilizado *aritmética modular* em vez da *aritmética real*. O conjunto de inteiros é calculado com a operação modular em função de um número primo p e esse conjunto de inteiros forma um **Corpo Finito** $\mathbb{F}$ onde a interpolação é possível. Assumindo que D é um valor inteiro é necessária a escolha de um número primo p maior que D e n. Os coeficientes $a_1, \ \ldots, a_{k-1}$ em $q(x)$ são escolhidos de maneira aleatória e uniformemente distribuídos através dos inteiros dentro de $[0, p)$ que por consequência estão dentro do corpo $\mathbb{F}$, e os valores $D_1, \ \ldots, D_n$ são calculados dentro do Corpo Finito $\mathbb{F}(\text{mod } p)$.

Com relação a recuperação do polinômio $q(x)$ através de interpolação polinomial existem algoritmos que atingem bons resultados com

complexidade $O(n \log^2 n)$ mas até algoritmos de complexidade quadrática ($O(n^2)$) conseguem bons resultados. Caso o valor numérico D seja um número muito grande que exija aritmética com precisão acima do que a máquina que está sendo usada oferece, o valor D pode ser quebrado em *blocos de bits* que precisam ser tratados separadamente. Esses blocos não podem ser arbitrariamente pequenos visto que o menor valor utilizável de p é $n + 1$ pois precisam existir ao menos $n + 1$ valores distintos no conjunto $[0, p)$ para calcular o polinômio $q(x)$. Essa limitação é contornável visto que operações modulares com dezesseis bits conseguem ser suficientes para aplicações com 64.000 D_i partes.

Por fim, quando comparamos o uso do *Esquema de Limiar* ou (k, n) *threshold scheme* com a solução de fechaduras e chaves é possível observar algumas propriedades:

1) O tamanho de cada parte D_i não excede o tamanho do dado D original

2) Quando k é mantido fixo, é possível adicionar ou deletar outras partes D_i sem interferir as outras partes D_i já existentes

3) É simples a mudança das partes D_1 sem a mudança do valor D original visto que é necessária apenas a mudança do polinômio $q(x)$ ($q'(x)$ por exemplo). Essa mudança pode acontecer de maneira frequente e consciente, aumentando a segurança e a confiabilidade visto que eventuais partes D_i comprometidas por brechas de segurança não serão mais válidas se não forem parte do novo conjunto de D'_i valores que geram o polinômio $q'(x)$

4) É possível criar um esquema de hierarquia dentro do conjunto de envolvidos deixando mais de uma parte D_i com a mesma entidade envolvida deixando que essa entidade possua um maior peso de decisão dentro do esquema

2.10 TRABALHOS RELACIONADOS

Algumas implementações foram consideradas no que se diz respeito a forma que foram realizadas e possíveis falhas de implementação. Esse estudo prévio veio como motivação para o desenvolvimento de uma ferramenta robusta e que comportasse um grau de Segurança e Confiabilidade superior das atuas propostas existentes.

Também foram consideradas algumas escolhas que melhorassem

os pontos falhos relacionados a essas ferramentas já disponíveis à comunidade.

2.10.1 libgfshare

Como primeiro trabalho relacionado e implementado por (SILVERSTONE; MCVITTIE, 2006) temos a biblioteca **libgfshare**, uma ferramenta que implementa o **Esquema de Segredo Compartilhado de Shamir** e está disponível no repositório oficial de softwares da distribuição Linux **Ubuntu**. A biblioteca pode ser obtida acessando o seguinte link:

`https://packages.ubuntu.com/cosmic/libgfshare-bin`

Ou pelo link do repositório de desenvolvimento

`https://git.gitano.org.uk/libgfshare.git/tree/`

Essa biblioteca tem como característica a utilização do Corpo de Galois GF(2**8) e oferece as funcionalidades de `split` e `join` com a utilização do Esquema de Shamir.

Ao fazer uma breve Análise do código, já encontramos alguns possíveis problemas como no trecho do arquivo `src/libgfshare.c` entre as linhas 46 a 56:

```
1  static void
2  _gfshare_fill_rand_using_random( unsigned char*
       buffer,
3                                   unsigned int count
                                        )
4  {
5    unsigned int i;
6    for( i = 0; i < count; ++i )
7      buffer[i] = (random() & 0xff00) >> 8;
8      /* apparently the bottom 8 aren't
9      * very random but the middles ones
10     * are
11     */
12 }
```

Existe um comentário que faz menção ao fato da geração de números aleatórios não parecem ser exatamente aleatórios em seus primeiros 8 dígitos. Essa função é utilizada para popular as variáveis de contexto no que diz respeito a números aleatórios e esse simples comentário talvez traga a tona o caráter de que essa biblioteca possui alguma falha de implementação ou possível brecha para ataques.

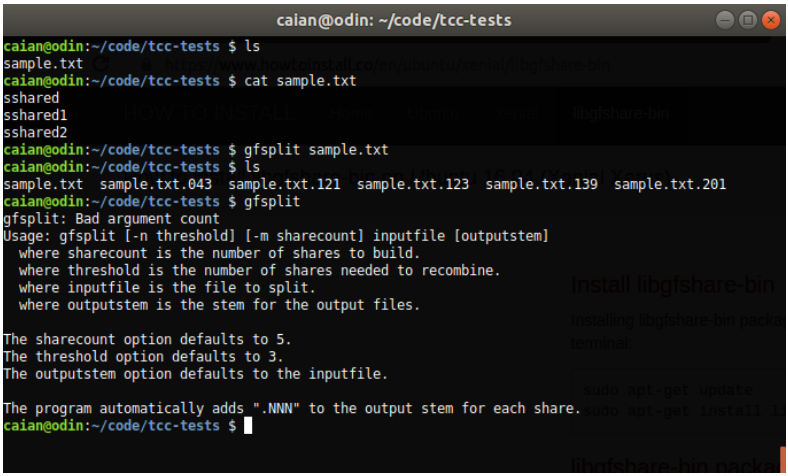

Figura 1 – Exemplo de Uso **libgfshare**

Ainda no que diz respeito ao uso da **libgfshare**, foram realizados alguns testes que serão mostrados e descritos nas Figuras a seguir. Entre os pontos que talvez possam gerar ainda mais brechas e que podemos ressaltar é o fato de que existe uma limitação no tamanho do Polinômio que pode ser gerado: $n = 6$. Enquanto podemos ter muito provavelmente pela representação de um arquivo de saída a criação de até $m = 999$, a limitação de um tamanho máximo de *shares* para a reconstrução do *Texto em Claro* necessários pode ser um fator crítico.

Iniciando os testes da ferramenta temos na Figura 1 que apresenta um arquivo de testes simples com o conteúdo também mostrado na Figura e um breve uso da ferramenta.

Podemos observar que o uso da ferramenta através do executável `gfsplit` faz como uso o valor padrão de 3 para o tamanho mínimo de *shares* necessários para a reconstrução do *Texto em Claro* e o valor padrão de 5 para a quantidade de *shares* gerados. É possível assumir então que o Polinômio gerado é um polinômio de terceiro grau.

Não foi possível determinar se a extensão do arquivo tem algo a ver com o valor que deve ser usado dentro do algoritmo para a reconstrução ou se é apenas um número aleatório que está sendo usado para evitar a colisão de nomes. Caso uma análise mais profunda revele essa relação, esse poderia ser um outro fator crítico de ataque ao algoritmo oferecido pela ferramenta.

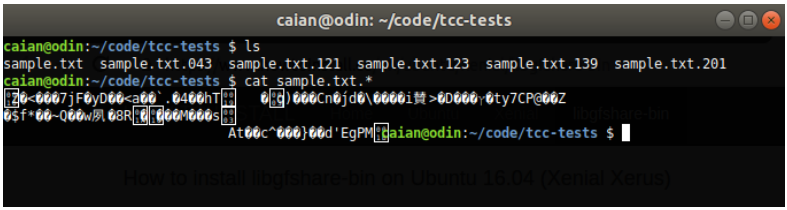

Figura 2 – Arquivos de *share* gerados pela **libgfshare**

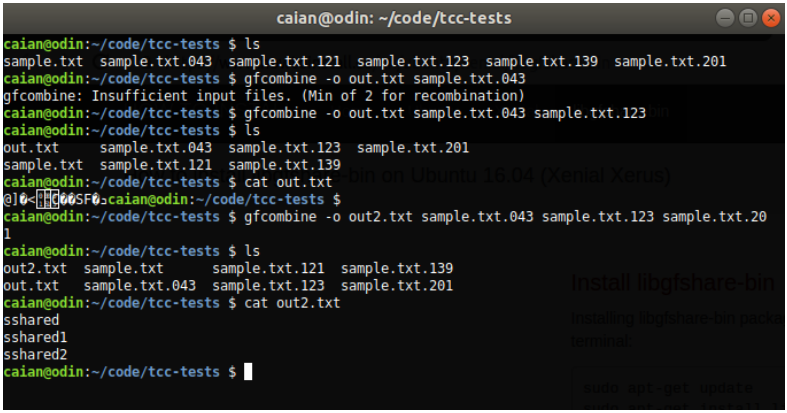

Figura 3 – Reconstrução do Arquivo Original pela **libgfshare**

Quanto a Figura 2, nela mostramos ao leitor o conteúdo interno dos arquivos de *shares* criados pela biblioteca *libgfshare*. Podemos ver que eles não possuem qualquer relação com o conteúdo original do arquivo `sample.txt`.

Na Figura 3 é possível observar que o executável `gfcombine` que irá reconstruir o segredo necessita de um mínimo de 2 partes para reconstruir e informa isso ao usuário do mesmo. É utilizado então dois arquivos para a reconstrução do arquivo original e é dado a ele o nome de `out.txt`. Em sequência o arquivo reconstruído é mostrado na tela e é possível observar que não existe relação aparente entre o arquivo `out.txt` e o arquivo original `sample.txt`.

Na sequência, é mostrada mais uma tentativa de reconstrução com o número de 3 arquivos e assim é possível realizar a reconstrução do arquivo original levando o nome de `out2.txt`. Seu conteúdo é

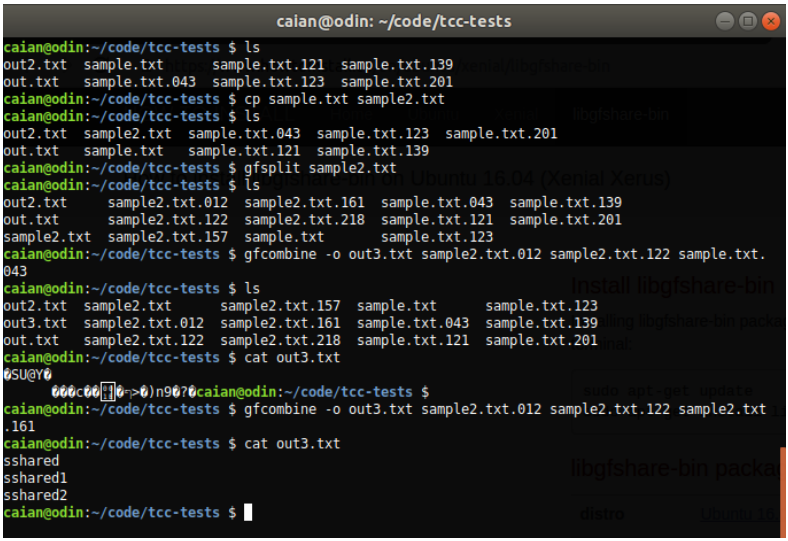

Figura 4 – Tentativa de reconstrução com **libgfshare**

exatamente igual ao conteúdo do arquivo original de nome `sample.txt`.

Na Figura 4 é possível observar uma tentativa de reconstrução do segredo usando *shares* de execuções distintas da ferramenta através do executável `gfsplit`. Ainda que o texto original fosse o mesmo a reconstrução utilizando arquivos de *share* das duas execuções não conseguiram reconstruir o segredo. Ao final é mostrado que a reconstrução do arquivo original é possível através do uso de *shares* de uma mesma rodada de produção dos *shares*.

Na Figura 5 é possível observar a limitação com relação ao tamanho do polinômio gerado. Essa limitação pode ser um fator limitante e crítico no caso do uso dessa biblioteca em uma aplicação real que precise de uma quantidade maior de *shares* mínimos para a reconstrução do *Texto em Claro*.

Ainda que a **libgfshare** seja uma ferramenta que já está como oferta ao usuário a um relativo tempo, ainda é possível observar algumas falhas críticas como a questão da geração de números aleatórios e a limitação do tamanho mínimo de *shares* necessários para a reconstrução do arquivo original.

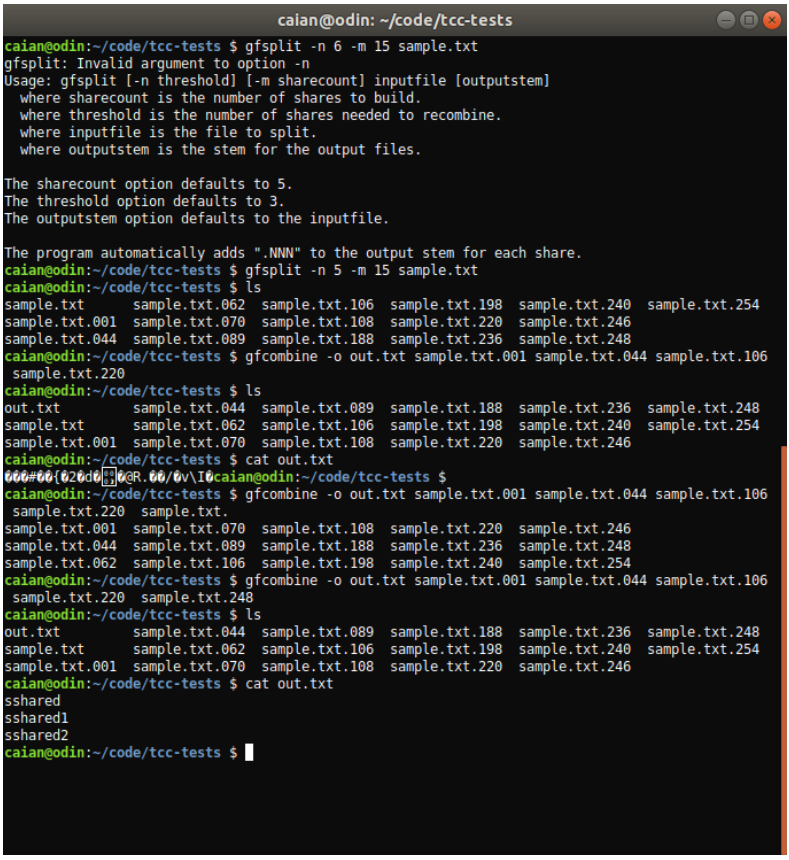

Figura 5 – Demonstração de Limitação da **libgfshare**

2.10.2 python-gfshare

Desenvolvida por Lamb (LAMB, 2017) de nome **python-gfshare**, a implementação tem como principal funcionalidade o que é chamado de *Wrapper*: a biblioteca **python-gfshare** encapsula as chamadas para a biblioteca discutida anteriormente e disponibiliza as chamadas para a mesma permitindo que uma aplicação implementada em `Python` possa realizar as operações de `split` e `join`.

Visto que essa implementação apenas encapsula as chamadas para a **libgfshare**, ela não será discutida a fundo. Basta declarar que os possíveis problemas, limitações e brechas encontradas na biblioteca que está sendo encapsulada ainda estão presentes internamente a **python-gfshare**.

Entretanto com a eventual melhoria ou correções da **libgfshare** a biblioteca em `Python` estará recebendo esses avanços de maneira integral.

2.10.3 Biblioteca sss

Desenvolvida por Daan Sprenkels e atualmente em sua versão 0.1.0 (SPRENKELS, 2017), essa biblioteca utiliza algumas premissas como

- Ser resistente a ataques do tipo *side-channel*
- Utilizar um MAC (*Message Authentication Code*) para garantir segurança ao segredo que está sendo compartilhado
- Produção de valores aleatórios durante sua execução através da própria plataforma que está executando a aplicação de maneira nativa

São premissas válidas e como o repositório declara, essa ferramenta pode ser utilizada no que ele mesmo declara como "mundo real", não sendo uma mera prova de conceito.

O autor também realiza um comentário sobre o fato de que o tamanho do segredo produzido é um pouco maior do que o dado original por motivos de segurança e ele adiciona uma camada a mais de segurança do que apenas a utilização do Segredo Compartilhado proposto por Shamir em seu trabalho. A biblioteca consegue cifrar um dado que possui um tamanho de 64 bytes e caso o usuário queira cifrar um *Texto*

em Claro de tamanho maior ele deve utilizar algum outro algoritmo para cifrar o *Texto em Claro* através de um algoritmo simétrico e cifrar a chave utilizada para realizar o procedimento e não o segredo.

Outro ponto importante é o fato de que o autor declara sua biblioteca segura contra ataques do tipo *side-channel* e contra Inviolabilidade (*Tamper-resistant*) durante a o uso da mesma. Ele ainda realiza uma comparação com outras bibliotecas existentes, inclusive a outra biblioteca citada neste trabalho, a **libgfshare** declarando a mesma insegura quanto a esses dois tipos de ataques. Apesar da declaração ser feita pelo autor, em nenhum momento foi descrita a forma pela qual os testes foram realizados nem o procedimento, entretanto é possível observar pelos comentários relacionados as análises que algum processo foi realizado para que esta afirmação fosse usada.

Quanto a geração de números aleatórios para o uso da biblioteca, é realizada de maneira nativa pela plataforma que ela está sendo executada, sendo que uma outra ferramenta desenvolvida pelo próprio autor da biblioteca é usada como referência: **randombytes** (SPRENKELS, 2016).

Como um primeiro fator limitante na biblioteca, não existe suporte a entrada e saída pela mesma sendo que um executável que oferece esse suporte foi criado usando a linguagem *Rust*. Existe também uma demonstração web, implementada também pelo autor. A biblioteca em si trata apenas da execução da operação de `split` e `join`, chamada pela API da biblioteca de `split` e `combine`.

Outro fator limitante ao observar o código é o fato de que com o uso de valores dentro da representação numérica da linguagem, a biblioteca também reside no mesmo Corpo da biblioteca **libgfshare** GF(2*8), trazendo uma limitação de tamanho máximo para o texto a ser cifrado por motivos de Segurança. O autor explica em seu próprio texto que um valor maior do que 64 bytes deve ser cifrado com um outro tipo de algoritmo como já foi descrito.

Entre os pontos não contemplados pela biblioteca, estão:

- Não há possibilidade a mudança do número primo que gera o Corpo
- O Corpo Finito usado é sempre o mesmo apesar de que os polinômios diferem entre si devido a geração de números aleatórios
- Não é gerado um novo polinômio para cada byte. O mesmo polinômio aleatório é utilizado para todos os bytes durante todas as rodadas de cifra

- Os valores aleatórios residem na implementação nativa da plataforma (SO) que está sendo utilizada

Apesar da biblioteca ter uma implementação notável que conta com o suporte a diversas operações lógicas de baixo nível e com garantias a ataques como o *side-channel*, ela implementa algumas funcionalidade extra para maiores garantias de Segurança como o uso de algoritmos para autenticação para suprir as limitações de implementação. É válido ainda mencionar que a biblioteca está em sua versão 0.1.0 e já possui seu uso com garantias e suporte superiores a outras alternativas existentes como o próprio autor menciona em sua documentação.

3 PROPOSTA

Relacionando a Fundamentação Teórica como base para que os conhecimentos aplicados específicos de Segredo Compartilhado fossem compreendidos foi pensado sobre o desenvolvimento de uma ferramenta que viesse sanar a necessidade de uma aplicação que fosse a implementação de Segredo Compartilhado não apenas como prova de conceito. Esse esforço foi então concretizado no resultado desse trabalho, a biblioteca **sshared**: uma aplicação que pode ser utilizada como biblioteca e vir a ser incluída em outras aplicações de maneira simples.

Este capítulo tem como propósito demonstrar questões como escolha de ferramentas para o desenvolvimento bem como a motivação de cada escolha de projeto. No primeiro momento serão discutidas as ferramentas utilizadas durante o desenvolvimento. A Arquitetura será definida e alguns diagramas serão mostrados auxiliando a descrição do funcionamento da biblioteca. Por fim a implementação será mostrada e comentada de maneira mais aprofundada com trechos de código.

Cabe neste momento também declarar que a escolha da linguagem C++ foi proposital e sua escolha será melhor discutida na seção de Implementação.

3.1 AMBIENTE E FERRAMENTAS

O desenvolvimento da biblioteca foi realizado com o uso de um ambiente Linux em uma distribuição Ubuntu de versão 18.04, sendo que todos os testes foram realizados nele. A detecção de vazamentos de memória (*memory leaks*) foi feito através da ferramenta *Valgrind* e a execução sequencial do código monitorada foi feita através da ferramenta *gdb - GNU Debugger*.

No que diz respeito a Compilação do código gerado, foi utilizada a ferramenta *Make* auxiliando o compilador *g++ - GNU C++ Compiler* onde um arquivo de *Makefile* foi produzido para os diferentes *targets*:

- *make*: compila a biblioteca e gera um executável para ser disponibilizado abertamente como *stand-alone*
- *make test*: compila a biblioteca e gera um executável de testes para ser testado
- *make static*: compila a biblioteca e gera um arquivo de biblioteca estática para ser incluído de forma estática em aplicações

- *make dynamic*: compila a biblioteca e gera um arquivo de biblioteca dinâmica para ser incluído de forma dinâmica em aplicações
- *make clean*: limpa o diretório de desenvolvimento de todos os arquivos compilados e gerados
- *make veryclean*: limpa o diretório de desenvolvimento de todos os arquivos compilados e gerados bem como os arquivos gerados pela execução dos testes

Com relação as operações matemáticas que são executadas tanto sobre um Corpo Finito quanto as operações polinomiais, optou-se pelo uso da biblioteca matemática **NTL**: Uma biblioteca matemática *Open-Source* e que possui um suporte robusto a todas as operações que viriam a ser realizadas.

Para a instalação dessa biblioteca e suas dependências, visando uma melhor configuração automatizada do ambiente de desenvolvimento e do uso da mesma pelo usuário final foi gerado um script de instalação e configuração da biblioteca **NTL** otimizando assim o tempo de configuração e setup do ambiente de desenvolvimento. Ele se encontra dentro do diretório de *scripts/*.

3.2 ORGANIZAÇÃO DE DIRETÓRIOS

Como premissa, a aplicação deveria ser extensível a outras implementações de **Segredo Compartilhado**. Dessa forma a organização da biblioteca deve ser robusta e extensível para que eventuais mudanças futuras sejam incentivadas mantendo o funcionamento das aplicações prévias funcionando em sua totalidade. A visualização dessa organização pode ser observada na Figura 6.

O diretório *include* possui todos os arquivos de headers que dizem respeito ao projeto como um todo. Ali estão contidos também eventuais arquivos de *templates* que serão utilizados pelo projeto. O diretório de *scripts* possui os scripts necessários para a instalação das ferramentas necessárias para o desenvolvimento como por exemplo a biblioteca **NTL** e suas dependências nos locais esperados pelo projeto.

O diretório *src* contém os arquivos de implementação referentes ao projeto em si e que dizem respeito ao funcionamento efetivo da biblioteca. Aos arquivos que dizem respeito a controle de iterações com o Sistema Operacional como abertura de arquivos ou arquivos que dizem respeito a estruturas de dados que serão utilizados de maneira horizontal por todo o projeto, estes estão concentrados no diretório *util*.

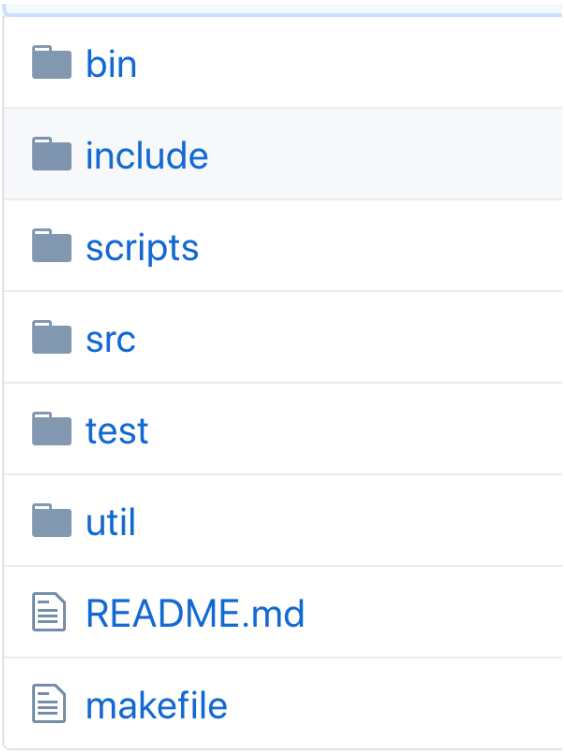

Figura 6 – Estrutura de Diretórios

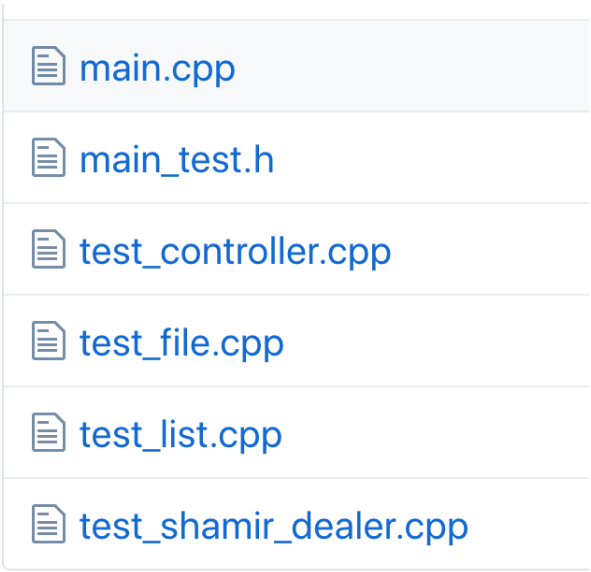

Figura 7 – Diretório de Testes

O diretório que contém os arquivos de testes nomeado de *test* deve ser auto-suficiente em relação ao resto do projeto e não deve depender de arquivos de headers ou de implementação misturados ao resto do projeto exceto ao módulo que ele está especificamente testando. Dessa forma, a organização interna desse diretório pode ser visualizada na Figura 7, e ele deve testar as funcionalidades de todas as partes do projeto de maneira mais atômica possível.

A implementação dos testes será descrita em detalhes porém é possível observar que cada arquivo será utilizado para testar um módulo específico do projeto, separadamente. Como exemplo, o arquivo *test_ shamir_ dealer.cpp* será responsável por verificar o funcionamento da interface do módulo *shamir_ dealer.cpp*.

3.3 ARQUITETURA

A Arquitetura da biblioteca foi estabelecida com tomadas de decisão baseadas nos conhecimentos do graduando e em discussões com o orientador sobre o comportamento que a biblioteca **sshared** deveria

ter. Essas decisões são embasadas na implementação do software que executa dentro do HSM e no módulo de acesso produzido pelo LabSEC em parceria com a empresa Kryptus e da implementação parcial do software do HSM no padrão PKCS#11 e do módulo de acesso a esse software, sendo o primeiro com participação direta do orientador e o segundo com participação conjunta do graduando.

A biblioteca possui uma interface de acesso a programas e chamadas externas que isola o programador usuário da mesma de modificações diretas em tempo de execução. Essa interface de acesso é o primeiro contato do programa executável que faz uso das funcionalidades, seja ele via acesso **estático**, **dinâmico** ou ***standalone***. Já dentro da biblioteca, os dados são verificados e a sequencia das operações são implementadas com o uso da biblioteca matemática **NTL**. Com as operações devidamente realizadas entre a biblioteca **sshared** e a biblioteca **NTL**, a tarefa requisitada pelo programa executável é concluída e o fluxo retorna ao programa para que ele siga realizando suas tarefas.

Esse fluxo pode ser observado na Figura 8 para a versão de acesso Dinâmico, na Figura 9 para a versão de acesso Estático e na Fiugra 10 para a versão de acesso *Standalone*. A diferença entre os modos de acesso é:

- **Acesso Dinâmico:** onde a Aplicação acessa a *sshared* através das bibliotecas disponíveis no Sistema Operacional
- **Acesso Estático:** onde a Aplicação acessa a *sshared* através da compilação da mesma junto ao programa aplicativo e ele carrega consigo uma biblioteca estática
- **Acesso *Standalone*:** onde é possível o acesso a *sshared* através de um executável da biblioteca funcionando como uma aplicação *standalone*

A ordem de chamadas pode ser observada em todas as Figuras de alto nível de acesso a biblioteca. A sequência se caracteriza por:

1. a *Aplicação* faz a requisição para a biblioteca **sshared** e passa o fluxo de execução para a biblioteca
2. para realizar suas operações, a **sshared** realiza chamadas para a biblioteca **NTL** e aguarda pelas operações
3. a biblioteca **NTL** realiza suas operações matemáticas e retorna o fluxo para a **sshared**

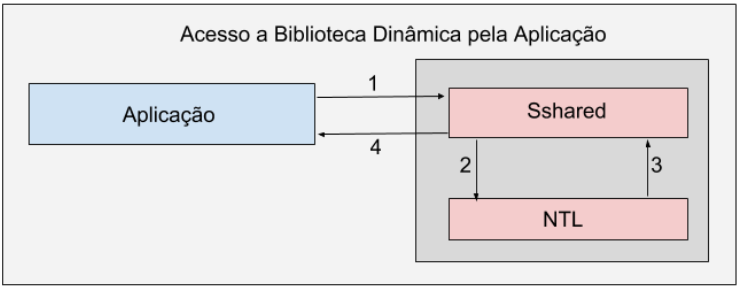

Figura 8 – Arquitetura Dinâmica

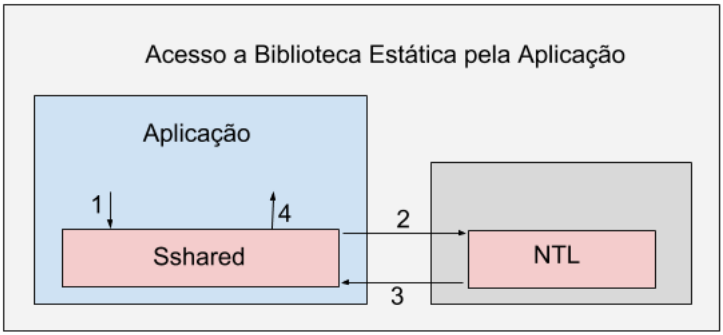

Figura 9 – Arquitetura Estática

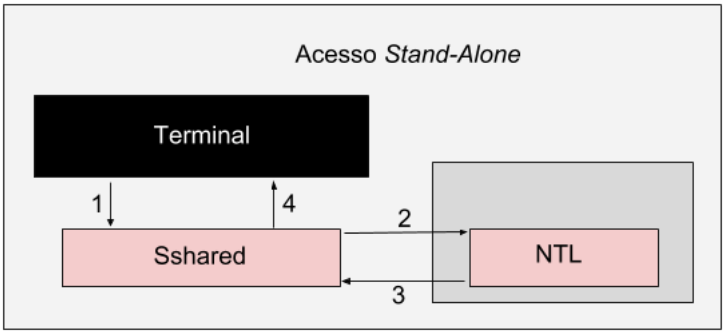

Figura 10 – Arquitetura *Standalone*

4. ao término do processo, a biblioteca **sshared** retorna o fluxo de execução para a aplicação

Com essa visão holística do funcionamento da biblioteca, é possível agora mostrar um Diagrama de Sequência que descreve o funcionamento de maneira menos abstrata ao leitor localizado na Figura 11. As chamadas de função não estão fiéis as chamadas realizadas pelas partes envolvidas e os valores passados como argumentos são valores que elucidam apenas as informações realmente importantes para a compreensão do funcionamento da biblioteca.

Ainda sobre a Figura 11, podemos comentar sobre os passos tomados pelas partes e enumerar alguns fatos interessantes

- Após a aplicação enviar o pedido de realizar uma divisão de um arquivo em partes, ele aguarda pelo término do processo que será realizado pelas outras partes envolvidas: **sshared**, **NTL** e a Abstração do Sistema de Arquivos do Sistema Operacional
- Baseado no tamanho do arquivo que foi lido, a biblioteca realiza a separação em tuplas do dado original para cada byte lido
- Um único polinômio é gerado **a cada iteração** de cada byte, com números de seus **coeficientes gerados de maneira aleatória** com o uso da biblioteca NTL
- Cada parte (*share*) de cada byte lido é instanciado com valores **aleatórios**

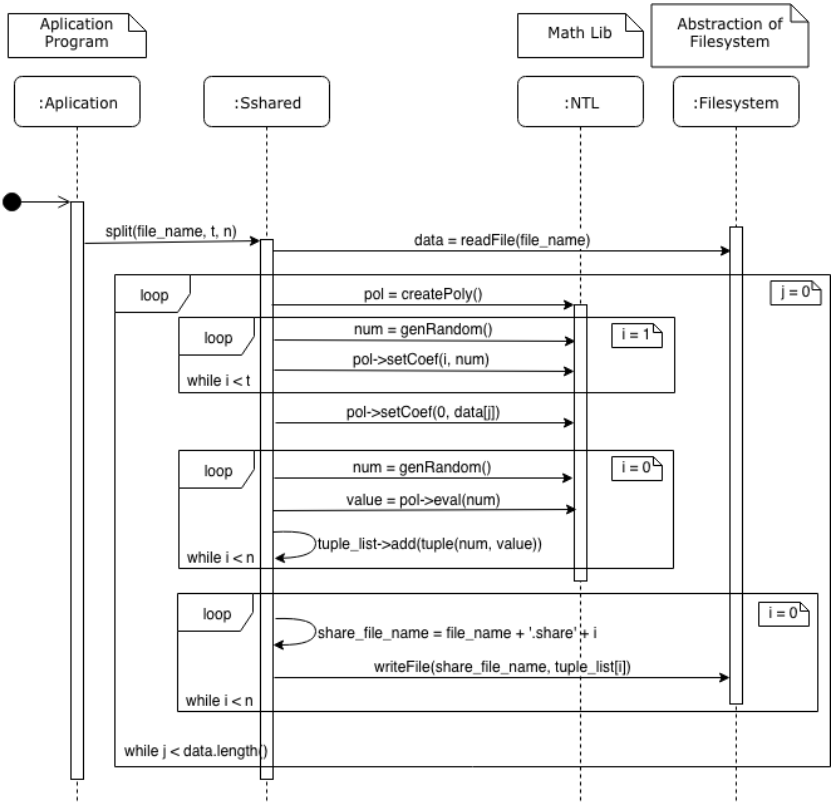

Figura 11 – *Split* - Visão da Aplicação

- Caso o arquivo possua apenas uma repetição de m bytes iguais onde $m = 20$, por exemplo, e sejam eles todos a letra 'a', as chances de existir um mesmo *share* que possa ser reutilizado no próximo byte são ínfimas visto que cada polinômio gerado para cada *share* é único e aleatório

Talvez um dos pontos mais importantes que pode ser ressaltado aqui é o fato de que para **cada byte** é gerado um polinômio único e aleatório. Isso garante que eventualmente o conhecimento de alguns valores de polinômios que revelem uma parte do *Texto em Claro* **não possam ser utilizados para revelar outra parte faltante** do *Texto em Claro*. Essa propriedade garante que um eventual comprometimento de **parte** significativa de alguns *shares* em quantidade mínima necessária para a recuperação do segredo daquela parte não resulte na recuperação **total** do segredo por algum tipo de técnica de Criptoanálise diferente da Força Bruta.

Com relação a função de recuperação do segredo, a Figura 12 traz o Diagrama de Sequência abstrato do funcionamento dessa funcionalidade em forma da visão da Aplicação. Novamente, as chamadas de função não estão fiéis as chamadas realizadas pelas partes envolvidas e os valores passados são apenas valores que facilitam a compreensão do funcionamento geral da biblioteca.

Neste Diagrama é possível observar que a Aplicação deve enviar uma lista de nomes de arquivos fazendo referência aos arquivos de saída que constituem os *shares* do arquivo original, chamada de *tuple_list*. Cada arquivo é lido e uma lista dessas tuplas é construída para guardar as tuplas de cada arquivo chamada de *evaluated_shares*. Após, é iterado sobre essa lista, *evaluated_shares* e os polinômios são interpolados. Com o polinômio interpolado para cada byte gerado, podemos recuperar o valor do byte original realizando a operação *eval(0)*. Então somamos cada byte a variável *result* e por fim realizamos a escrita do arquivo de saída com o *Texto em Claro* recuperado.

3.3.1 Formato de Arquivo `.share`

O estabelecimento de um formato de saída para a funcionalidade de `split` e de entrada da funcionalidade de `join` foi pensado de maneira simples o suficiente para que fosse de fácil compreensão e de fácil utilização.

Ele recebeu o nome de extensão de `.share` e leva ao final de seu nome o valor sequencial da ordem que foi gerado. Essa ordem não

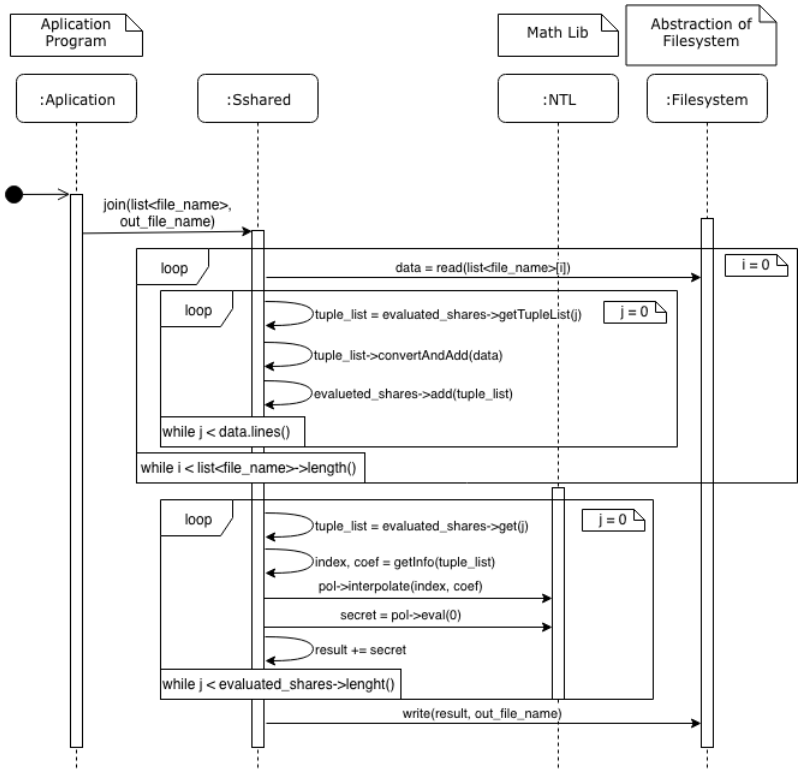

Figura 12 – *Join* - Visão da Aplicação

possui qualquer relação com a forma que o dado é operado nem com a ordem necessária de inclusão de arquivos no momento de reconstrução do segredo, tendo apenas esse número como um valor sequencial para uma melhor contagem da quantidade de arquivos gerados e evitar a colisão de nomes entre os mesmos.

O formato interno desse arquivo deve respeitar um padrão simples com as seguintes premissas:

- Uma **linha** representa uma **tupla**
- Uma **tupla** possui um separador para os valores
- Os valores da tupla sempre são valores numéricos e sempre respeitam a seguinte ordem

$$x, q(x)$$

- A ordem de construção das linhas é exatamente a ordem de caracteres do arquivo original. Como exemplo: o primeiro caractere se torna a primeira linha, o segundo caractere se torna a segunda linha e assim subsequentemente

Trazendo como exemplo, um arquivo do tipo `.share` gerado pela biblioteca tem o seguinte formato

```
1 8813,37279
2 89145,47697
3 42236,1066
4 48739,29926
5 79761,89605
6 8077,53250
7 10478,28198
```

É possível observar que o separador escolhido foi o caractere de vírgula e que os valores numéricos não possuem qualquer relação com o arquivo original a não ser a quantidade de linhas sendo a mesma quantidade de caracteres.

3.4 IMPLEMENTAÇÃO

Desde sua concepção a biblioteca deveria dar suporte a escalabilidade e a utilização através das diversas plataformas e Sistemas Operacionais. A implementação na linguagem C++ foi feita como uma

alternativa a outras linguagens consideradas e que ofereceu algumas facilidades que foram mais valorizadas em detrimento de outras. Entre elas, podemos citar:

- Existência de uma biblioteca para operações matemáticas com otimizações e implementação *thread safe* que extraem do hardware a execução com o menor *overhead* possível

- Uma vez compilado, é possível sua utilização sem a necessidade de uma Máquina Virtual ou de um Interpretador como outras linguagens exigem

- Possibilidade de inclusão em outros executáveis como uma biblioteca estática, fazendo com que a dependência final seja apenas a biblioteca externa NTL

- Possibilidade de inclusão em outros executáveis como uma biblioteca dinâmica, diminuindo assim o *overhead* de inclusão em mais de um programa executável do mesmo código e diminuindo o tamanho dos executáveis finais que utilizarão a biblioteca trazendo como dependência final também a biblioteca NTL

- Possibilidade de criação de um *script* de instalação com as versões específicas que estão sendo usadas pela biblioteca de suas dependências, sem a necessidade de criação de um ambiente virtual de desenvolvimento, do uso de contêineres de execução de código isolado ou de dependências de atualizações de bibliotecas de terceiros

- Possibilidade de inclusão em executáveis de outras linguagens visto que existem alternativas para a inclusão de códigos em C/C++ em Java e Python, por exemplo

Não obstante, a escolha da linguagem também foi baseada na escolha de suas limitações e para tal podemos citar:

- Tratamento de memória manual e inexistência de implementação nativa da linguagem de um *Garbage Colletctor*

- Dependência de uma biblioteca matemática que quando instalada ocupava uma grande quantidade de espaço em disco sendo esta tendo quase 2GB de tamanho após completamente instalada

3.4.1 Namespace

Primeiramente é importante documentar o fato de que foi criado um `namespace` para o projeto, fazendo referência ao nome da bibliteca: `SS`. É possível observar que todos os arquivos header e os arquivos de implementação possuem referências a esse artifício da linguagem como mostra o trecho a seguir

```
1  #ifndef DEALER_H
2  #define DEALER_H
3
4  // ...
5
6  namespace SS
7  {
8      // ...
9      class Dealer {
10     public:
11         virtual ~Dealer() { }
12         virtual TupleList* split(std::string data,
               unsigned int t, unsigned int n) = 0;
13         virtual std::string join(TupleList* shares)
               = 0;
14     };
15 }
16 #endif
```

O motivo de uso de `namespaces` é simples: Evitar a colisão de nomes tanto de variáveis quanto de nomes. Visto que já dentro da própria implementação temos duas classes implementando métodos `split()`, é possível deduzir que no caso de uso da biblioteca em programas maiores venham a ter nomes que possam colidir entre si. O *overhead* de escrita é apenas utilizar a especificação de que estamos usando a variável ou a chamada de um método interno a biblioteca colocando `SS::` antes de cada uso, como podemos observar na implementação de um método a seguir

```
1  std::string SS::FileHandler::read(std::string from)
       {
2      SS::ReadableFile* in_file = new SS::
           ReadableFile(from);
3      in_file->open();
4      std::string data = in_file->read();
```

```
5       in_file->close();
6       delete in_file;
7       return data;
8   }
```

3.4.2 Escolhas e Padrões de Projeto

As escolhas de projeto também levaram em conta o fato de conceitos como *Single Responsability* e *Dependency Inversion Principle* que incentivam questões como a separação de responsabilidades e a dependência de abstrações e não de instâncias. Esses conceitos poderão ser melhores observados na análise do código que será feita na sequência mas de maneira abstrata um pequeno trecho de código pode demonstrar como esses conceitos foram aplicados.

```
1   class Controller {
2   public:
3       bool filter_message(const char* mes[], int size
            );
4   private:
5       Dealer* dealer;
6       void split();
7       void join();
8   }
```

É possível observar que apesar do `Controller` ser o controlador geral de uma chamada para a biblioteca ele não realiza as operações diretamente com a biblioteca **NTL**: Sua responsabilidade é escolher o que o programa deve fazer baseado na entrada que o usuário requisitou. Dessa forma observamos também a dependência de um `Dealer` que não sabemos ao certo qual **Esquema de Segredo Compartilhado** estará sendo implementado e só saberemos a respeito disso quando o usuário requisitar alguma operação.

Operações de criação, abertura, leitura e escrita em arquivos também foram encapsuladas em uma abstração chamada `File` e baseado no tipo de arquivo que será operado essa escolha acontece apenas quando o código está sendo executado. Para auxiliar em operações mais específicas de leitura e escrita, foi criado também uma abstração chamada `FileHandler`, que possui funções estáticas e utilizam as implementações da abstração `File` internamente.

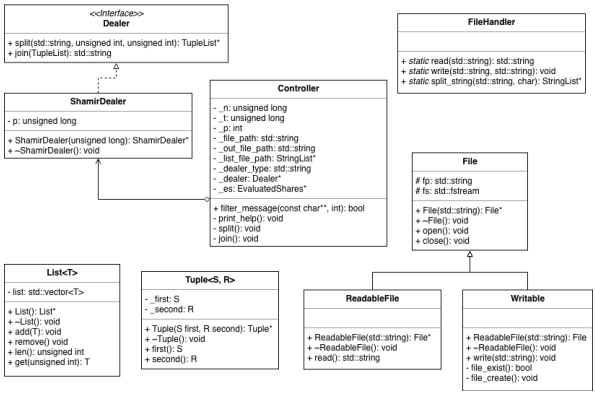

Figura 13 – Implementação

3.4.3 Diagrama de Classes

O Diagrama de Classes pode ser visualizado na Figura 13. Nela podemos ver a interface que deve ser implementada pelos *Dealers* como foi realizado com o `Dealer` referente ao esquema de Segredo Compartilhado de Shamir representado na classe `ShamirDealer`. Essas classes e a classe controladora da biblioteca se encontram dentro do diretório *src* pois são parte integrante do sistema de maneira única.

Também podemos visualizar questões referente aos arquivos que serão abertos, lidos ou escritos e que devem estender a classe `File`, podendo criar arquivos somente de leitura com a classe `ReadableFile` ou de leitura e escrita, como está implementado na classe `WritableFile`. Esses dois tipos foram implementados dentro de um arquivo e se encontram dentro do diretório *util* do projeto, junto a implementação da classe `FileHandler`.

Com relação aos dados que devem ser guardados sequencialmente não importando o tipo primitivo ou a classe a qual pertencem, foi criado uma classe que encapsula a classe que pertence a biblioteca padrão do C++ e leva o nome de *Standard Vector*. Essa classe implementada recebeu o nome de `List` e sua implementação aconteceu diretamente no arquivo de header, posicionado dentro do diretório de includes.

Foi criado também uma estrutura de dados para guardar uma Tupla de dados com 2 valores genéricos. Essa classe levou o nome de `Tuple` e ela é utilizada em vários locais da aplicação facilitando a

forma que os dados são armazenados. Assim como a classe `List` ela foi criada com o auxílio de templates de C++ e apenas serão criadas implementações em tempo de compilação para os tipos de dados que são usados na aplicação e não há necessidade de escrever o código de cada tipo de dado que será utilizado na tupla.

Com relação as outras estruturas de dados que não possuem implementações visto que são apenas definições de tipos que facilitam a escrita, leitura e o desenvolvimento, elas estão devidamente declaradas em arquivos que são utilizadas. Para uma compreensão futura do código escrito e mostrado, elas serão apresentadas na sequência de maneira completa:

```
1  // Value x of q(x)
2  typedef std::string ssX;
3  // Value of q(x) based on x
4  typedef std::string ssY;
5  // Tuple definition
6  typedef Tuple<ssX, ssY> ShareTuple;
7  // List of shares
8  typedef List<ShareTuple> TupleList;
9  // List of Evaluated Shares as TupleLists
10 typedef List<TupleList*> EvaluatedShares;
11 // List of Strings
12 typedef List<std::string> StringList;
```

3.4.4 Classe `Controller`

Sobre a implementação da classe controladora da biblioteca chamada de `Controller`, por ter um contato direto com o exterior ela possui várias propriedades e funções de set que são usadas com base na única função pública: `filter_message()`. Podemos observar a implementação dessa função realizando a verificação de diversas premissas no trecho de código a seguir.

```
1  bool SS::Controller::filter_message(const char* mes
       [], int size) {
2      // initial check
3      if(size < 2) {
4          std::cerr << "[sshared] Error: Number of
               arguments incorrect. Use -h (help)" <<
               std::endl;
```

```
5           return false;
6       }
7
8       // print help
9       if(mes[1][1] == 'h') {
10          this->print_help();
11          return true;
12      }
13
14      // check if split or join was passed
15      std::string sj = mes[1];
16      if(sj.compare("split") != 0 && sj.compare("join
            ") != 0) {
17          std::cerr << "[sshared] Error: Pass the
                operation (split or join). Use -h (help
                )" << std::endl;
18          return false;
19      }
20      int nsize = size - 1;
21
22      // check for number of arguments
23      if(nsize % 2 == 0 || nsize < 2) {
24          std::cerr << "[sshared] Error:  Number of
                arguments incorrect. Use -h (help)" <<
                std::endl;
25          return false;
26      }
27      // ...
28  }
```

Nesse trecho de código é possível observar a verificação da entrada passada para valores incorretos ou que não respeitam o padrão estabelecido comum de passagem de parâmetros a programas aplicativos de terminal: uma *flag* e o seu valor ou seus valores. Também é possível observar a existência de uma função de ajuda ao usuário que está utilizando a biblioteca pela primeira vez caso esteja utilizando a versão *Standalone*: `print_help()`. Pode ser observado seu uso na Figura 14.

Mais a frente, ma mesma função é possível observar os valores sendo setados internamente e a execução de uma função: *split* ou *join*.

```
1   bool SS::Controller::filter_message(const char* mes
```

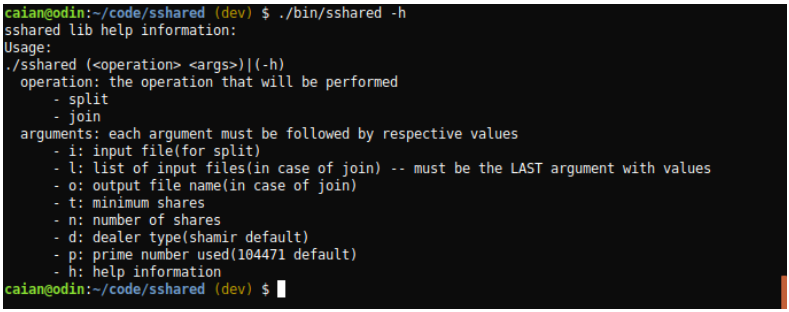

Figura 14 – Execução da função `print_help()`

```
    [], int size) {
2       // ...
3       // set values on Controller
4       for(int i = 2; i < nsize; i += 2) {
5           if(mes[i][1] == 'l') {
6               if(!this->set_list_filepath(mes, i,
                    size - i)) {
7                   std::cerr << "[sshared] Error:
                        Number of arguments incorrect.
                        Use -h (help)" << std::endl;
8                   return false;
9               }
10              break;
11          }
12          if(!this->set_value(mes[i], mes[i+1])) {
13              return false;
14          }
15      }
16
17      // do split/join operation
18      if(sj.compare("split") == 0) {
19          this->split();
20      } else if (sj.compare("join") == 0) {
21          this->join();
22      } else {
23          return false;
24      }
```

```
25      return true;
26  }
```

Todas as *flags* disponíveis e questões referentes ao funcionamento serão melhores descritas na seção de **Utilização**. Por hora, basta declarar que as funcionalidades disponíveis internas a biblioteca são encapsuladas na classe `Controller` pelas funções `split()` e `join()`.

Ainda sobre a classe Controladora é importante comentar que ela realiza as chamadas ao objeto `ShamirDealer` que encapsula tanto o método de `split()` quanto de `join()` referentes a implementação de Segredo Compartilhado de Shamir. Ela é instanciada com o uso de um número para a criação do **Corpo Finito** e é importante que esse número seja um número **primo** superior ao tamanho do dado que será cifrado. Foi escolhido que caso o usuário não envie nenhum número como parâmetro o número usado será 104471.

As chamadas aos métodos do objeto `ShamirDealer` ocorrem internamente para cada byte de um arquivo para a operação de `split()` ou da lista de arquivos no caso da operação de `join()`. A implementação do método `Controller::split()` pode ser observada abaixo fazendo uso tanto do objeto `ShamirDealer`. Caso seu funcionamento precise de uma melhor compreensão é válido citar o Diagrama de Sequência existente na Figura 11 que elucida o funcionamento de maneira abstrata.

```
1  void SS::Controller::split() {
2      if(this->_dealer_type.empty()) {
3          std::cerr << "[sshared] Error: No Dealer
               Type defined" << std::endl;
4      }
5      if(this->_dealer_type.compare("shamir") == 0) {
6          // read file
7          std::string data = SS::FileHandler::read(
               this->_file_path);
8
9          this->_dealer = new ShamirDealer(this->_p);
10         this->_es = new EvaluatedShares();
11         SS::TupleList* tl;
12
13         // step 1 - split each byte
14         for(unsigned int i = 0; i < data.length() -
               1; i++) {
15             tl = this->_dealer->split(data.substr(i
```

```
                , 1), this->_t, this->_n);
16              this->_es->add(tl);
17          }
18
19          // step 2 - save to files
20          for(unsigned int i = 0; i < this->_n; i++)
                {
21              std::string out_share = "";
22              // write the Shares on each file
23              for(unsigned int j = 0; j < this->_es->
                    len(); j++) {
24                  tl = this->_es->get(j);
25                  SS::ShareTuple st = tl->get(i);
26                  out_share += st.first() + "," + st.
                        second() + "\n";
27              }
28
29              std::string share_file_name = this->
                    _file_path + ".share" + std::
                    to_string(i);
30              SS::FileHandler::write(out_share,
                    share_file_name);
31          }
32
33          // cleanup
34          for (unsigned int i = 0; i < this->_es->len
                (); i++) {
35              delete this->_es->get(i);
36          }
37          delete this->_es;
38          delete this->_dealer;
39          return;
40      }
41      std::cerr << "[sshared] Error: Dealer Type not
            supported" << std::endl;
42  }
```

O uso do método `FileHandler::read()` que será explicado mais a frente, acontece após algumas verificações iniciais e realiza a leitura de um arquivo e o retorno do mesmo em formato de `std::string`. Em sequência na linha 9 a criação do objeto `ShamirDealer` com o uso do número primo passado é feita. Esse será o `Dealer` utilizado no processo

interno. É também instanciado o atributo do tipo `EvaluatedShares`, `this->_es` que fica responsável de guardar os *shares* em forma de tuplas e também é declarada uma lista de tuplas `TupleList* tl` que ficará responsável por guardar o ponteiro durante a iteração do método ao realizar a operação de `ShamirDealer::split()` de cada byte do arquivo. Esse processo pode ser observado nas linhas 14 até a linha 17.

Nesse ponto, com o uso da função `ShamirDealer::split()` é possível observar o fato de que a cada iteração desse loop é criado um novo **polinômio** para cada byte que está sendo operado. As chances de um mesmo polinômio se repetir aqui é muito baixa. As chances de um mesmo polinômio ser usado para o mesmo byte que está repetido no *Texto em Claro* são ainda menores.

A frente, nas linhas 20 a 30 são realizadas operações para criação de arquivos e escrita dos *shares* nesses arquivos. Não há muito o que comentar a não ser o uso da função `FileHandler::write()` que será explicada a frente e a construção do arquivo no formato de um arquivo de *share*.

Quanto a função `Controller::join()`, podemos observar no trecho de código a seguir sua implementação e relembrar a Figura 12 que demonstra o Diagrama de Sequência dessa operação de maneira abstrata. Assim como o código gerado para a funcionalidade anterior, este código possui alguns comentários que facilitam a compreensão do que está sendo feito a cada momento.

```
1  void SS::Controller::join() {
2      if(this->_dealer_type.empty()) {
3          std::cerr << "[sshared] Error: No Dealer
               Type defined" << std::endl;
4      }
5      if(this->_dealer_type.compare("shamir") == 0) {
6
7          this->_es = new EvaluatedShares();
8
9          // step 1 - open files and get the shares
10         for(unsigned int i = 0; i < this->
               _list_file_path->len(); i++) {
11             // read the shares
12             std::string data = SS::FileHandler::
                   read(this->_list_file_path->get(i))
                   ;
13
14             SS::StringList* lstring = SS::
```

```
                FileHandler::split_string(data, '\n
                ');
15
16              for(unsigned int j = 0; j < lstring->
                    len(); j++) {
17                  SS::TupleList* tl;
18                  // check for tuples on es
19                  if(this->_es->len() == 0 || j >=
                        this->_es->len()) {
20                      tl = new SS::TupleList();
21                      this->_es->add(tl);
22                  }
23                  // build the tuples
24                  std::string line = lstring->get(j);
25                  SS::StringList* elements = SS::
                        FileHandler::split_string(line,
                         ',');
26                  SS::ssX x = elements->get(0);
27                  SS::ssY y = elements->get(1);
28                  SS::ShareTuple st(x, y);
29                  tl = this->_es->get(j);
30                  tl->add(st);
31                  delete elements;
32              }
33              delete lstring;
34          }
35
36          // step 2 - join the shares
37          this->_dealer = new ShamirDealer(this->_p);
38          std::string out = "";
39
40          for(unsigned int i = 0; i < this->_es->len
                (); i++) {
41              SS::TupleList* tl = this->_es->get(i);
42              out += this->_dealer->join(tl);
43          }
44
45          SS::FileHandler::write(out, this->
                _out_file_path);
46
47          // cleanup
```

```
48          for (unsigned int i = 0; i < this->_es->len
                (); i++) {
49              delete this->_es->get(i);
50          }
51          delete this->_dealer;
52          delete this->_es;
53          return;
54      }
55      std::cerr << "[sshared] Error: Dealer Type not
            supported" << std::endl;
56  }
```

É possível notar na implementação o uso da lista de nomes de arquivos representada pela variável na linha 10 `this->_list_file_path`. Essa variável é iterada e a lista que contém os *shares* é remontada até que todos os arquivos passados sejam inseridos nessa lista. O uso do método `FileHandler::split_string()` é feito na linha 14 e será explicado mais a frente mas por hora basta conhecer que ele é usado para quebrar o arquivo em linhas e mais a frente na linha 26 para também quebrar cada linha entre os valores respectivos dos *shares*.

Já com a lista de *shares* reconstruída e ainda não sabendo se ela será suficiente para a reconstrução do *Texto em Claro*, é instanciado o objeto `ShamirDealer` para que a recuperação do segredo seja realizada. Esse processo pode ser observado nas linhas 37 a 43.

Por fim, o método `FileHandler::write()` é utilizado para escrever o arquivo de saída com o arquivo recuperado e a memória é liberada. Como não cabe a biblioteca conhecer ou comparar o arquivo recuperado ao arquivo original, não é realizada nenhuma verificação a respeito dessa funcionalidade e é deixado ao programa aplicativo que utiliza a biblioteca a verificação.

3.4.5 Classe `ShamirDealer`

Entrando mais a fundo sobre a funcionalidade essencial a biblioteca, chegamos ao objeto que fará uso da biblioteca **NTL** e realizará as operações matemáticas internamente. Essa classe estende da classe abstrata `Dealer` e possui internamente a implementação da proposta de **Segredo Compartilhado de Shamir** que foi descrita anteriormente no trabalho. Na sequência, será mostrado o *header* com as assinaturas do objeto em questão.

```
1 #include <NTL/ZZ_p.h>
```

```
2  #include <NTL/ZZ_pX.h>
3  #include "dealer.h"
4
5  namespace SS
6  {
7      class ShamirDealer : public Dealer {
8      public:
9          ShamirDealer(unsigned long prime) : p(prime
               ) { }
10
11         TupleList* split(std::string data, unsigned
               int t, unsigned int n);
12         std::string join(TupleList* shares);
13
14     private:
15         unsigned long p;
16     };
17 }
```

Notamos o uso de um atributo do tipo `unsigned long` para guardar o número primo. Esse tipo de dado garante que apenas números positivos serão usados e aumenta a representação numérica em relação ao tipo de dado `int` ou até mesmo `long` em detrimento de sua representação não considerar o primeiro bit para representação do sinal do número.

Com relação a biblioteca **NTL**, os *headers* específicos referentes a representação numérica de números dentro de um Corpo Finito e a representação de um polinômio com seus coeficientes dentro de um Corpo Finito são respectivamente os arquivos `ZZ_p.h` e `ZZ_pX.h`. A representação numérica padrão é incluída por esses *headers*, não sendo assim necessária a sua inclusão. Caso seja de interesse ao leitor procurar maiores informações, os *headers* específicos das representações de números e polinômios da matemática dos números Inteiros são os arquivos `ZZ.h` e `ZZX.h`, respectivamente.

A assinatura dos métodos são descritas também nesse *header* para que sejam implementados conforme a interface pede que seja feito.

Com o conhecimento das assinaturas dos métodos e do uso do objeto `Controller`, é possível agora mostrar a implementação do método responsável pela quebra do arquivo original, `ShamirDealer::split()`.

```
1  SS::TupleList* SS::ShamirDealer::split(std::string
       data, unsigned int t, unsigned int n) {
```

```
2
3       // Error check
4       if(t == 0) {
5           std::cerr << "[sshared] Error: t value
                invalid" << std::endl;
6           return NULL;
7       }
8       if(n == 0) {
9           std::cerr << "[sshared] Error: n value
                invalid" << std::endl;
10          return NULL;
11      }
12      if(n < t) {
13          std::cerr << "[sshared] Error: n value
                invalid(n<t)" << std::endl;
14          return NULL;
15      }
16
17      SS::TupleList* rv = new SS::TupleList();
18
19      // Init ZZ_p
20      NTL::ZZ prime = NTL::conv<NTL::ZZ>(this->p);
21      NTL::ZZ_p::init(prime);
22
23      NTL::ZZ_pX pol;
24
25      const char * data_char = data.c_str();
26      unsigned long d = static_cast<unsigned long>(
            data_char[0]);
27
28      for(unsigned long i = 1; i < t; i++) {
29          NTL::ZZ_p coef = NTL::random_ZZ_p();
30          NTL::SetCoeff(pol, i, coef);
31      }
32      NTL::SetCoeff(pol, 0, d);
33
34      for(unsigned int i = 0; i < n; i++) {
35          NTL::ZZ_p val = NTL::random_ZZ_p();
36          SS::ssY eval = std::to_string(NTL::conv<
                long>(NTL::eval(pol, val)));
37          SS::ssX index = std::to_string(NTL::conv<
```

```
                long>(val));
38          SS::ShareTuple tuple(index, eval);
39
40          rv->add(tuple);
41      }
42
43      return rv;
44  }
```

Após algumas verificações iniciais acerca da validação dos valores passados, a implementação instancia um número inteiro responsável por ser o valor que vai caracterizar o Corpo Finito como é possível observar nas linhas 20 e 21. Em sequência é declarado o polinômio dentro do Corpo e como a biblioteca **NTL** salva o contexto do número primo que está sendo usado para operar o Corpo, não há necessidade de novamente informar o valor.

A seguir, nas linhas 25 e 26 é realizada a conversão do valor numérico do byte que será operado. Dentro do *loop* das linhas 28 a 31 o polinômio é preenchido com valores aleatórios dentro do Corpo Finito com o uso de uma função de que gera números aleatórios da biblioteca **NTL** `NTL::random_ZZ_p()` e o valor é inserido na posição específica do polinômio.

Na linha 32 algo importante acontece: é inserido o valor numérico do byte que será operado no polinômio no termo independente. É esse o valor que queremos guardar e que vamos recuperar quando o polinômio for reconstruído.

No segundo *loop* entre das linhas 34 a 41 os *shares* são criados baseados em um número aleatório dentro do Corpo, novamente com o uso da função geradora de números aleatórios da biblioteca **NTL**, e o polinômio é avaliado com esse valor. Essa Tupla de valores é criada como *índice* e *valor* e pode ser observada na linha 38. Na linha 40 ela é inserida na lista de tuplas que será retornada ao final da execução do método.

Para o método `ShamirDealer::join()`, o código será apresentado na sequência e com ele é possível recuperar o segredo caso o número mínimo de *shares* seja fornecido. Caso contrário, o polinômio recuperado não será o mesmo que realizou a operação descrita anteriormente e o segredo não poderá ser recuperado.

```
1  std::string SS::ShamirDealer::join(SS::TupleList*
       shares) {
2
```

```
3       // Error check
4       if(!shares) {
5           std::cerr << "[sshared] Error: no shares
                passed" << std::endl;
6           return NULL;
7       }
8
9       // Init ZZ_p
10      NTL::ZZ prime = NTL::conv<NTL::ZZ>(this->p);
11      NTL::ZZ_p::init(prime);
12
13      // create the vectors of coefficients and index
14      NTL::Vec<NTL::ZZ_p> coef;
15      NTL::Vec<NTL::ZZ_p> ind;
16
17      // sets length of vectors
18      coef.SetLength((long) shares->len());
19      ind.SetLength((long) shares->len());
20
21      for(long i = 0; i < shares->len(); i++) {
22          ind[i] = NTL::conv<NTL::ZZ_p>(std::stoul(
                shares->get(i).first()));
23          coef[i] = NTL::conv<NTL::ZZ_p>(std::stoul(
                shares->get(i).second()));
24      }
25
26      // recreate the polynomial
27      NTL::ZZ_pX pol;
28
29      // interpolate to find the polynomial
30      pol = NTL::interpolate(ind, coef);
31
32      NTL::ZZ_p zero_index;
33      zero_index = NTL::conv<NTL::ZZ_p>((long) 0);
34      int result = NTL::conv<int>(NTL::eval(pol,
            zero_index));
35      char result_char = static_cast<char>(result);
36      std::string eval(1, result_char);
37
38      return eval;
39  }
```

Nota-se bastante semelhança em chamadas de função da biblioteca **NTL** entre a implementação do método `ShamirDelaer::split()` e `ShamirDelaer::join()`. a parte inicial em que o Corpo Finito é construído nas linhas 10 e 11 e quando o polinômio é instanciado na linha 27 o código é praticamente o mesmo. Entre as linhas 11 e 27 entretanto é utilizada uma lista que a biblioteca **NTL** oferece e pode ser observado nas linhas 14 e 15 como `NTL::Vec<NTL::ZZ_p>`. Elas ficarão responsáveis por guardar os dados referentes aos *coeficientes* e aos *índices* que foram passados do polinômio que está sendo reconstruído garantindo que eles sejam devidamente guardados em memória. Esses vetores tem seus tamanhos definidos nas linhas 18 e 19 e são preenchidos no *loop* entre as linhas 21 e 24.

Na linha 30 temos a chamada para a função que é responsável por reconstruir o polinômio dentro da biblioteca **NTL** e que necessita do envio dos coeficientes e dos índices no tipo de listas implementadas pela própria **NTL**. Esse é o principal motivo pelo qual não foi utilizada a lista implementada pelo graduando para guardar esses valores. Com essa função chamada, um polinômio será reconstruído e será guardado na variável `pol`. Basta então a avaliação do polinômio com o coeficiente 0 para que o segredo seja recuperado caso o polinômio reconstruído seja efetivamente o mesmo polinômio usado para a quebra do segredo em *shares*. Isso pode ser observado nas linhas 32 a 34.

Nas linhas subsequentes o valor numérico é convertido ao tipo desejado de retorno `std::string` e retornado.

3.4.6 Classe `FileHandler`

Internamente a classe `Controller` foi necessária a criação de uma camada acima das classes que já implementam uma iteração básica com arquivos pela biblioteca. Essa abstração ficou responsável por ler arquivos e inserir em um `std::string` bem como escrever uma `std::string` a um arquivo. Essa classe levou o nome de `FileHandler` e está declaradas abaixo fazendo uso internamente das classes que estendem a classe `File`: `ReadableFile` e `WritableFile`.

```
1  std::string SS::FileHandler::read(std::string from)
       {
2      SS::ReadableFile* in_file = new SS::
           ReadableFile(from);
3      in_file->open();
4      std::string data = in_file->read();
```

```
5       in_file->close();
6       delete in_file;
7       return data;
8   }
9
10  void SS::FileHandler::write(std::string content,
        std::string to) {
11      SS::WritableFile* out_file = new SS::
            WritableFile(to);
12      out_file->open();
13      out_file->write(content);
14      out_file->close();
15      delete out_file;
16  }
```

Essa abstração consegue limpar o código dentro das funções `Controller::split()` e `Controller::join()` no que dizem respeito a legibilidade. Em ambas, o objeto é instanciado para que na sequência seja aberto pelo método `File::open()`. A operação em questão é realizada seja ela ler de um arquivo ou escrever em um arquivo, o arquivo é fechado com o uso do método `File::close()` e a memória é liberada com `delete` e caso o arquivo seja lido, é retornado o valor de leitura do mesmo em formato `std::string`.

Questões internas como a existência do arquivo ou criação de arquivo para o caso de escrita são realizadas internamente pela abstração `File` e não é de conhecimento da classe `FileHandler` caso algum erro venha a ocorrer.

Em auxílio a esses dois métodos e vendo que a sua utilização estava essencialmente atrelada ao uso de arquivos, foi também criada uma abstração para que um `std::string` fosse quebrada em partes baseado em um caractere. Esse método levou o nome de `split_string()` e pode ser obervado na sequência. Ele foi utilizado para quebrar um `std::string` que contém a leitura em uma lista de `std::string`, ou como foi definido um tipo para esse dado: `StringList`. Também foi utilizado para quebrar uma linha do arquivo de *share* que possui uma tupla com os valores $(x, q(x))$ referente as avaliações do polinômio.

```
1   SS::StringList* SS::FileHandler::split_string(std::
        string data, char delimiter) {
2       std::stringstream ssdata(data);
3       std::string token;
4       SS::StringList* lstring = new SS::StringList();
```

```
5       while(std::getline(ssdata, token, delimiter)) {
6           lstring->add(token);
7       }
8       return lstring;
9   }
```

3.4.7 Testes

Auxiliando a Implementação, foram desenvolvidos testes da funcionalidades da biblioteca e das partes mais críticas de código. Esses testes podem ser observados no diretório *test* e utilizando o *target* `make test` podemos compilar esses testes e ver os mesmos através do executável `stest`. A Figura 15 traz essa sequência de chamadas e baseado no arquivo `sample2.txt` que contém em seu interior um texto simples são gerados arquivos nos formatos de *share* de maneira sequencial. Vale ressaltar mais uma vez que a numeração ao final de cada arquivo faz referência apenas a ordem que os mesmos foram escritos e nada tem a ver com os valores ou polinômios ali dentro guardados.

O arquivo `sample2.txt` tem em seu interior o seguinte texto

```
1  sshared
2  sshared1
3  sshared2
```

Como exemplo, o arquivo `sample2.txt.share0` será mostrado na sequencia

```
1   54804,2107
2   20320,57247
3   80357,93613
4   98292,100737
5   76593,41184
6   13056,16453
7   104023,100542
8   39786,65027
9   39229,31520
10  35494,3124
11  26607,97586
12  93575,103755
13  91345,58785
14  98368,40109
15  24704,12243
```

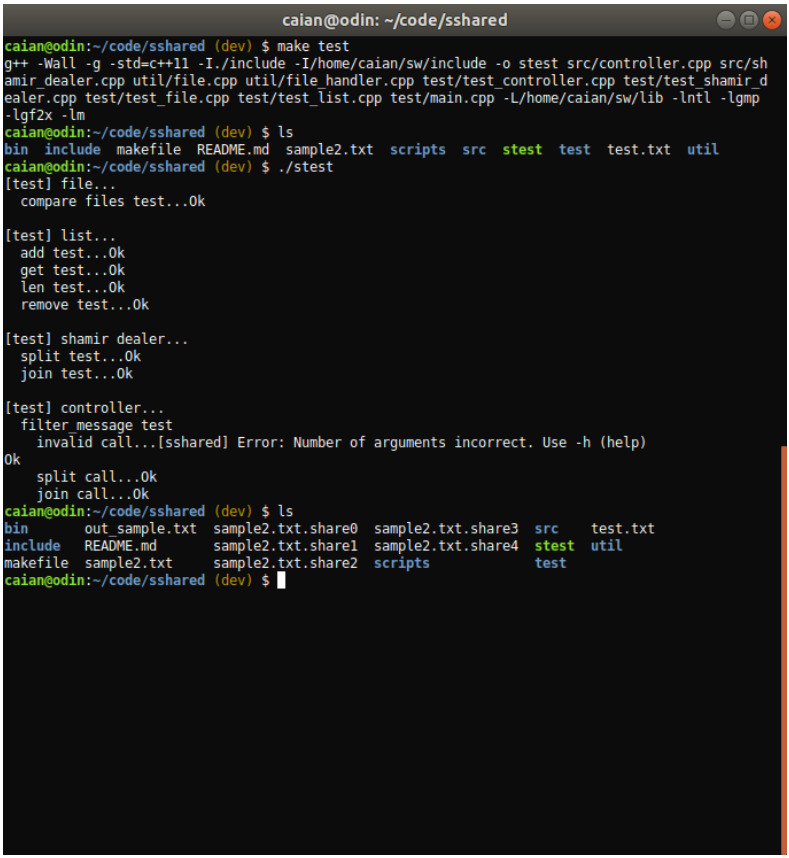

Figura 15 – Execução dos Testes

```
16  88583,13121
17  40581,13596
18  84761,56694
19  12264,8768
20  80848,97162
21  46792,9208
22  7692,59449
23  83309,70875
24  47069,20187
25  3554,28814
```

Com pouco esforço é possível observar uma das premissas da biblioteca: As duas letras iniciais do arquivo são o caractere s e as duas primeira linhas do arquivo de *share* tem como tuplas dois valores sem qualquer relação entre si: $54804, 2107$ e $20320, 57247$, demonstrando que o valor 0 no nome do arquivo não tem relação com o valor usado como coeficiente no polinômio e que os dois valores não fazem qualquer referência ao fato de que os dois primeiros caracteres do arquivo são o caractere s.

3.5 UTILIZAÇÃO E EXEMPLOS

Como descrito previamente, a biblioteca **sshared** possui *targets* diferentes dependendo do tipo de uso que a aplicação ou o usuário final desejar realizar. Em suma, existem 3 (três) modos de compilação para a execução do usuário final. São eles:

- **Standalone**: O usuário utiliza a biblioteca como um executável através da execução via linha de comandos
- **Estático**: O usuário utiliza a biblioteca como uma parte de seu programa e o seu executável leva consigo o código já compilado como arquivos objeto. A biblioteca deve ser passada como um argumento no processo de compilação
- **Dinâmico**: O usuário utiliza a biblioteca com acesso da mesma através da disponibilidade da mesma entre as bibliotecas de Sistema Operacional, sendo necessária que a mesma seja *linkada* no momento de compilação

Apesar do modo de utilização como biblioteca ser parecido, o uso da biblioteca na sua forma **Estática** possui o *overhead* de que o

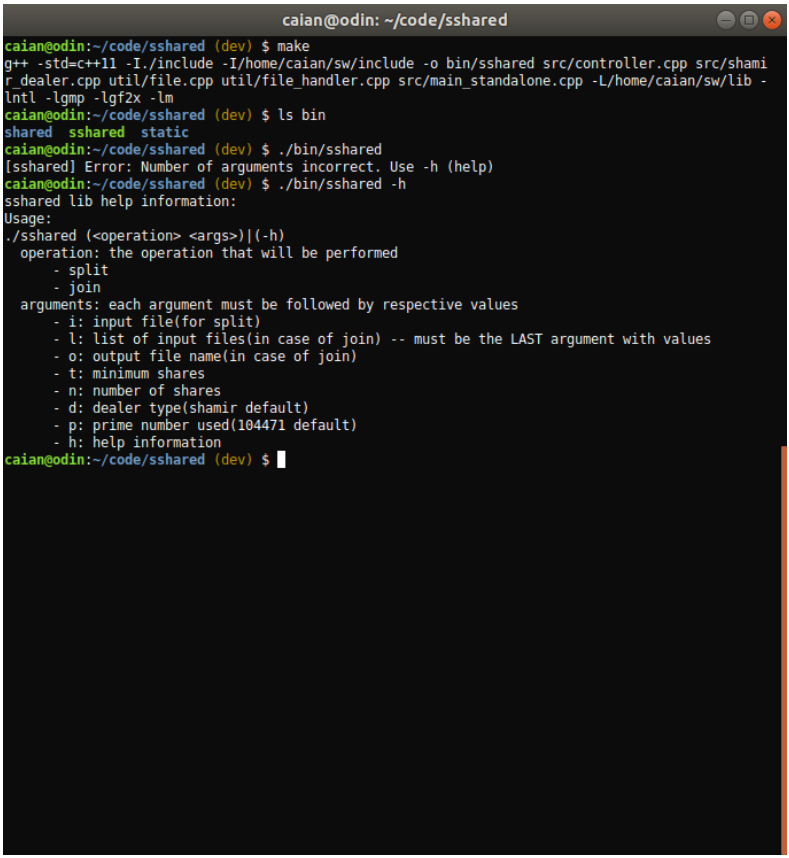

Figura 16 – Execução do *Standalone*

programa executável terá consigo o código da biblioteca internamente aumentando o tamanho do mesmo. Eventuais mudanças de código na biblioteca, melhorias e correções de bugs acarretam na necessidade de compilação do programa executável com o código atualizado. Por outro lado, a disponibilidade da biblioteca pelo Sistema Operacional não será necessária e o programa aplicativo funciona com a única dependência da biblioteca **NTL** externa.

Por outro lado, caso exista uma versão da biblioteca em sua forma **Dinâmica** oferecida pelo Sistema Operacional através da instalação da mesma na lista de bibliotecas do mesmo o programa executável do usuário não precisará ter consigo o código da biblioteca. Entre os benefícios, uma mudança, correção de bug ou melhoria na **sshared** pode ser observada sem a necessidade de uma nova compilação do programa executável.

Visto que a biblioteca **NTL** é utilizada de maneira **Dinâmica** pela **sshared**, resta ao usuário final escolher a melhor forma de montar seu processo de compilação baseado no uso final. As possibilidades oferecidas pela **sshared** dão ao mesmo total liberdade com relação a forma que o usuário irá incluir a biblioteca em seu aplicativo.

3.5.1 Executável *Standalone*

Visando facilitar a compreensão, a utilização da biblioteca será feita em seu formato *Standalone*, como pode ser observado na Figura 16. A seguir, teremos o trecho de código que representa a função `main()` contida dentro do arquivo `main_standalone.cpp`. Ela é responsável por receber a entrada do executável e tratar a mesma.

```
1  #include <iostream>
2  #include <string>
3
4  #include "controller.h"
5  #include "shamir_dealer.h"
6
7  #include "readable_file.h"
8  #include "writable_file.h"
9
10
11 int main(int argc, const char* argv[]) {
12
13     SS::Controller* con = new SS::Controller();
```

```
14      con->filter_message(argv, argc);
15
16      // cleanup
17      delete con;
18      return 0;
19  }
```

Os parâmetros passados como entrada no uso da biblioteca como programa executável *Standalone* deve respeitar o padrão de uso mostrado na Figura 16. Esse padrão garante o uso correto da mesma e em caso de parâmetros errados uma mensagem de erro será mostrada na tela ao usuário para que ele consulte a maneira correta que ele deve utilizar a aplicação.

Na Figura 17 observamos o que poderia vir a ser um uso real da biblioteca após sua compilação. Ela é uma captura de tela de um terminal realizando operações dentro do diretório de desenvolvimento.

Nessa figura, observamos a primeira linha onde a biblioteca é compilada em modo *Standalone* com o comando `make`. Na sequencia o arquivo `sample2.txt` é copiado para dentro do diretório `bin`, onde se localiza o executável. O conteúdo desse arquivo é

```
1  sshared
2  sshared2
3  sshared3
```

O comando seguinte já é realizando uma operação de *split* da biblioteca e pode ser observado o seu uso da biblioteca na Figura 14. Esse comando trata de chamar a biblioteca como um executável e passar a ela os parâmetros:

- `split`: o operação a ser realizada
- `-i`: o parâmetro que informa sobre o próximo parâmetro ser um argumento de input
- `sample2.txt`: o arquivo que será usado como entrada
- `-t`: o parâmetro que informa sobre o próximo parâmetro ser um argumento de quantidade mínima de *shares* necessária para a reconstrução do segredo
- `5`: o valor referente a quantidade mínima de *shares* necessária para a reconstrução do segredo
- `-n`: o parâmetro que informa sobre o próximo parâmetro ser um argumento de quantidade total de *shares* que será gerada

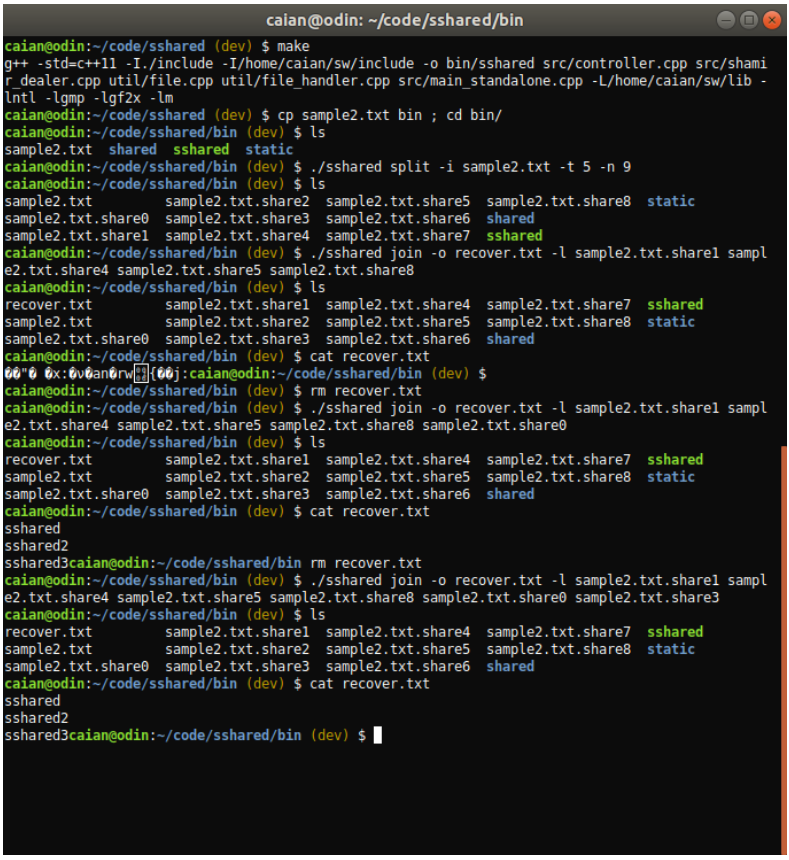

Figura 17 – Exemplo de Uso

Esse comando deve produzir 9 arquivos com a extensão `.share` e isso é mostrado com o comando seguinte listando o conteúdo do diretório. Esses arquivos respeitam o padrão de arquivos de *share* guardando os valores como tupla $x, q(x)$.

O comando seguinte é novamente um pedido ao executável para que ele reconstrua o arquivo original. Os parâmetros passados são:

- `join`: o operação a ser realizada
- `-o`: o parâmetro que informa sobre o próximo parâmetro ser um argumento de output
- `recover.txt`: o nome arquivo que será usado como saída
- `-l`: o parâmetro que informa sobre os próximos parâmetros serem a lista de nomes de arquivos que serão utilizados como entrada para a produção do arquivo de saída
- `*.share`: nomes de arquivos nesse formato que serão usados como entrada para a reconstrução do arquivo original

Especificamente sobre esse comando, ele está sendo usado de maneira **correta** mas não deve ser capaz de reconstruir o segredo (arquivo original) de maneira correta. O motivo disso é o fato de que está sendo passado a ele uma quantidade menor que a necessária para a reconstrução do segredo.

Quando o comando de `share` foi executado, foi pedido uma separação em 9 arquivos(*shares*) e que fossem necessários um mínimo de 5 *shares* para que o segredo fosse reconstruído. Nesse comando, o arquivo reconstruído com o nome de `recover.txt` é mostrado com o comando `cat recover.txt` e seu conteúdo é observado ser diferente do arquivo original, `sample2.txt`.

Na linha seguinte, o arquivo `recover.txt` é excluído e novamente é feita uma chamada de reconstrução do segredo, agora com a quantidade mínima necessária exata de 5 *shares* e o arquivo é capaz de ser reconstruído em sua totalidade. Isso pode ser observado na chamada seguinte, mostrando na tela o arquivo `recover.txt`.

Na sequência, é enviado um novo comando `join` com uma quantidade maior do que a mínima necessária e o arquivo também é reconstruído corretamente.

Algo interessante e importante de se observar é o fato de que não há qualquer necessidade de envio sequencial dos *shares* ou quais *shares* estão sendo passados ao programa desde que os mesmos sejam gerados a partir do arquivo original e respeitem os polinômios.

4 ANÁLISE

A **Análise** do trabalho acontecerá de forma dividida entre dois tópicos maiores: **Análise de Modelagem** e **Análise de Segurança**.

Enquanto a primeira parte irá tratar especificamente sobre a análise de código, desenvolvimento e testes da aplicação relacionados ao código, a segunda parte irá ficar responsável por considerar análises de Segurança e no uso da biblioteca de forma mais geral e de uma visão de usuário.

Com relação aos **Testes**, já foi relatado em seções anteriores a existência dos mesmos e esse será o momento em que os mesmos serão relatados com alguma profundidade e para a compreensão do leitor. Questões de utilização da biblioteca podem ser observadas mais a fundo neste ponto. Auxiliado pelo conhecimento do formato de arquivo de `share` declarado previamente e que caracteriza uma das partes do segredo que está sendo compartilhado, eles serão explorados e mostrados no que toca a questões de Segurança.

4.1 MODELAGEM

Primeiramente é importante citar que os testes são compilados e eles utilizam a biblioteca e suas partes em pontos que poderiam gerar algum tipo de erro ou falha de implementação. O target dentro do arquivo de `makefile` é `make test` e um executável de testes é gerado. A execução desses testes se dá de maneira a verificar sequencialmente os objetos internos e a abstração que controla a biblioteca quando a mesma é compilada em modo *Standalone*. A função principal do executável de testes possui as chamadas de cada função de testes e será apresentada a seguir e sua execução já foi relatada no trabalho na Figura 15

```
1  /* main tests include */
2  #include "main_test.h"
3
4  // main test function
5  int main(int argc, char* argv[]) {
6
7      std::cout << "[test] file... " << std::endl;
8      test_file();
9
10     std::cout << std::endl;
```

```
11      std::cout << "[test] list... " << std::endl;
12      test_list();
13
14      std::cout << std::endl;
15      std::cout << "[test] shamir dealer... " << std
            ::endl;
16      test_shamir_dealer();
17
18      std::cout << std::endl;
19      std::cout <<"[test] controller... " << std::
            endl;
20      test_controller();
21
22      return 0;
23  }
```

Essa função trata apenas de chamar as funções internas e interagindo com o output padrão de maneira a mostrar o que estará sendo testado.

Seu arquivo de header é apresentado agora e nele contém as declarações das funções que devem ser implementadas pelos outros arquivos. É nesse arquivo que devem ser inseridos novas declarações de testes, caso elas sejam implementadas por trabalhos futuros.

```
1  #ifndef MAIN_TEST_H
2  #define MAIN_TEST_H
3
4  #include <iostream>
5  #include <string>
6
7  #include "controller.h"
8  #include "shamir_dealer.h"
9  #include "readable_file.h"
10 #include "writable_file.h"
11 #include "list.h"
12
13 /* controller tests */
14 void test_controller();
15 /* list tests */
16 void test_list();
17 /* file tests */
18 void test_file();
```

```
19 /* shamir_dealer tests */
20 void test_shamir_dealer();
21
22 #endif
```

As funções que testam a implementação do objeto `List` não serão mostradas em sua totalidade visto que não fazem parte dos conteúdos envolvidos na implementação direta do trabalho. Uma breve demonstração de implementação dos testes do objeto `List` é o código que está na sequência. Visto que o objeto faz uso de `Templates`, os testes realizam operações dom dados dos tipos `int` e `std::string`. Os testes completos podem ser encontrados no arquivo `test/text_list.cpp`.

```
1  /* main tests include */
2  #include "main_test.h"
3
4  /* list tests */
5  void test_list() {
6      // SS::List creation
7      SS::List<int>* il = new SS::List<int>();
8
9      // TEST: add test
10     bool check = true;
11     std::cout << "  add test...";
12
13     int el0 = 3;
14     il->add(el0);
15     if(il->get(0) != el0) check = false;
16
17     if(!check) {
18         std::cout << "Error on int" << std::endl;
19         delete il;
20         return;
21     }
22     std::cout << "Ok" << std::endl;
23
24     // TEST: get test
25     std::cout << "  get test...";
26
27     if(il->get(0) != el0) check = false;
28
29     if(!check) {
```

```
30          std::cout << "Error on int" << std::endl;
31          delete il;
32          return;
33      }
34      std::cout << "Ok" << std::endl;
35
36      // TEST: len test
37      std::cout << "  len test...";
38
39      if(il->len() != 1) check = false;
40      if(!check) {
41          std::cout << "Error on int" << std::endl;
42          delete il;
43          return;
44      }
45      std::cout << "Ok" << std::endl;
46
47      // TEST: remove test
48      std::cout << "  remove test...";
49
50      il->remove();
51      if(il->len() != 0) check = false;
52      if(!check) {
53          std::cout << "Error on int" << std::endl;
54          delete il;
55          return;
56      }
57      std::cout << "Ok" << std::endl;
58
59
60      // cleanup
61      delete il;
62      return;
63  }
```

Esse arquivo relata o ciclo de vida de uma lista de inteiros em sua totalidade:

1. É criada a lista
2. Elementos são inseridos
3. Seu tamanho é verificado

4. Elementos são removidos

5. Novamente seu tamanho é verificado

6. A memória é liberada

Já sobre os objetos que implementam a abstração de arquivos `File`, os testes ocorrem dentro do arquivo `test/test_file.cpp` e o mesmo contém o seguinte trecho de código

```
1  /* main tests include */
2  #include "main_test.h"
3
4  /* file tests */
5  void test_file() {
6
7      // file name
8      std::string fp = "test.txt";
9
10     // writable file
11     SS::WritableFile* wf = new SS::WritableFile(fp)
           ;
12     wf->open();
13     std::string wfbuf = "Test file.\n1 2 3 test";
14     wf->write(wfbuf);
15     wf->close();
16     delete wf;
17
18     // readable file
19     SS::ReadableFile* rf = new SS::ReadableFile(fp)
           ;
20     rf->open();
21     std::string rfbuf = rf->read();
22     rf->close();
23     delete rf;
24
25     bool check = true;
26     // TEST: compare test
27     std::cout << "  compare files test...";
28     if(rfbuf.compare(wfbuf) == 0) check = true;
29     else check = false;
30     if(!check) {
```

```
31          std::cout << "\nError on compare" << std::
                endl;
32          return;
33      }
34
35      std::cout << "Ok" << std::endl;
36
37  }
```

Os testes se caracterizam pela manipulação de arquivos, passando pelo ciclo de vida de ambos os objetos que abstraem um arquivo: `ReadableFile` e `WritableFile`:

1. É aberto um arquivo (ou criado caso o mesmo não exista)
2. O arquivo é escrito
3. O arquivo é fechado
4. O arquivo é aberto novamente em formato de Leitura
5. É lido seu conteúdo
6. O arquivo é fechado
7. É comparado o conteúdo escrito no arquivo pela primeira parte com o conteúdo lido na segunda parte

Após os testes sobre componentes de uso da biblioteca, os testes sobre as funcionalidades principais da mesma serão demonstradas. Primeiro teremos a funcionalidade oferecida pela abstração que implementa um `Dealer`: o objeto `ShamirDealer`. Segue o arquivo `test/test_shamir_dealer.cpp`:

```
1  /* main tests include */
2  #include "main_test.h"
3
4  // Mock values
5  static const unsigned long p = 104471;
6  static const std::string data = "a";
7
8  // static const ShareTuple share_0("0", "97");
9  static const SS::ShareTuple share_1("1", "111");
10 static const SS::ShareTuple share_2("2", "187");
11 static const SS::ShareTuple share_3("3", "379");
```

```
12 static const SS::ShareTuple share_4("4", "741");
13
14 void test_split_shamir_dealer() {
15
16     std::cout << "  split test...";
17
18     SS::ShamirDealer* sd = new SS::ShamirDealer(p);
19     bool check = true;
20
21     SS::TupleList* tl;
22     unsigned int t = 3;
23     unsigned int n = 5;
24     tl = sd->split(data, t, n);
25
26     if(!tl) check = false;
27     if(tl->len() != n) check = false;
28
29     // cleanup
30     if(!!tl) delete tl;
31     if(!!sd) delete sd;
32
33     if(!check) {
34         std::cout << "Error" << std::endl;
35         return;
36     }
37
38     std::cout << "Ok" << std::endl;
39 }
40
41 void test_join_shamir_dealer() {
42
43     std::cout << "  join test...";
44
45     SS::ShamirDealer* sd = new SS::ShamirDealer(p);
46     bool check = true;
47
48     SS::TupleList* tl = new SS::TupleList();
49     // Match the known values from split
50     // the secret must not be known
51     // tl->add(share_0); // 'a' = 97
52     tl->add(share_1);
```

```
53      tl->add(share_2);
54      tl->add(share_3);
55      tl->add(share_4);
56      std::string rv = sd->join(tl);
57
58      if(rv.empty()) check = false;
59      if(rv.compare(data) != 0) check = false;
60
61      // cleanup
62      if(!!tl) delete tl;
63      if(!!sd) delete sd;
64
65      if(!check) {
66          std::cout << "Error" << std::endl;
67          return;
68      }
69
70      std::cout << "Ok" << std::endl;
71  }
72
73  /* shamir_dealer tests */
74  void test_shamir_dealer() {
75      test_split_shamir_dealer();
76      test_join_shamir_dealer();
77  }
```

A parte final do arquivo possui uma separação em duas funções que encapsulam as funcionalidades oferecidas pelo objeto `ShamirDealer`: `split` e `join`.

Especificamente sobre a funcionalidade de `Split`, ela realiza a separação do segredo em uma única Lista de *Tupla* visto que a separação em partes ocorre utilizando apenas um único caractere. O caractere escolhido é a letra *a* e esse segredo deverá ser separado nessa Lista de *Tuplas* que deverá ter obrigatoriamente o tamanho da variável *n* passada a chamada do método `split()`. Para a reconstrução, uma quantidade mínima de 3 tuplas é necessária basado no valor *t* passado. Após essa verificação de tamanho da lista a memória é limpa e a função é retornada ou encerrada.

Quanto a funcionalidade de `Join`, foi realizado um cálculo de separação de segredo e foram estabelecidas as *Tuplas* que estão criadas no começo do arquivo. É sabido que essas tuplas reconstroem o Segredo e precisamos testar se o método `join()` consegue realizar esse tipo

de operação. As *Tuplas* são adicionadas a uma Lista de *Tuplas* e a mesma é usada para reconstrução do segredo. Após é testado se o valor de retorno não é vazio e se ele realmente contém o valor original, o caractere *a*.

Após essa análise dos testes de funcionalidades isoladas, podemos avançar para o teste do controlador do executável *Standalone* que também faz parte dos objetos oferecidos pela **sshared** ao usuário final: o objeto `Controller` através dos testes contidos dentro do arquivo `test_controller.cpp`

```
1  /* test header include */
2  #include "main_test.h"
3
4  static const int argc_error = 1;
5  static const char* argv_error[argc_error] = {""};
6
7  static const int argc_split = 10;
8  static const char* argv_split[argc_split] = {
9      "sshare",
10     "split",
11     "-i",
12     "sample2.txt",
13     "-t",
14     "3",
15     "-n",
16     "5",
17     "-p",
18     "104471"
19 };
20
21 static const int argc_join = 12;
22 static const char* argv_join[argc_join] = {
23     "sshare",
24     "join",
25     "-p",
26     "104471",
27     "-o",
28     "out_sample.txt",
29     "-l",
30     "sample2.txt.share0",
31     "sample2.txt.share1",
32     "sample2.txt.share2",
```

```
33      "sample2.txt.share3",
34      "sample2.txt.share4"
35  };
36
37  void test_filter_message() {
38
39      std::cout << "  filter_message test";
40
41      SS::Controller* con = new SS::Controller();
42      bool check = true;
43
44      std::cout << std::endl;
45      std::cout << "    invalid call...";
46      check = con->filter_message(argv_error,
            argc_error);
47      if (check) {
48          std::cout << "Error" << std::endl;
49      } else {
50          std::cout << "Ok" << std::endl;
51      }
52      delete con;
53      con = new SS::Controller();
54
55      std::cout << "    split call...";
56      check = con->filter_message(argv_split,
            argc_split);
57      if(!check) {
58          std::cout << "Error" << std::endl;
59      } else {
60          std::cout << "Ok" << std::endl;
61      }
62      delete con;
63      con = new SS::Controller();
64
65      std::cout << "    join call...";
66      check = con->filter_message(argv_join,
            argc_join);
67      if(!check) {
68          std::cout << "Error" << std::endl;
69      } else {
70          std::cout << "Ok" << std::endl;
```

```
71      }
72
73      // cleanup
74      delete con;
75  }
76
77  /* controller tests */
78  void test_controller() {
79
80      test_filter_message();
81
82  }
```

Ao começo do arquivo é possível observar que temos algumas entradas que estão sendo usadas para validar as funcionalidades. nas linhas 4 e 5 observamos que existem variáveis que serão usadas em uma chamada da biblioteca de maneira incorreta e deverá ser mostrado na tela um aviso sobre o uso da mesma e como conseguir visualizar todas as funcionalidades através da chamada do método interno de ajuda, `print_help`.

Nas linhas 7 a 35 são declarados as variáveis que serão usadas pelos testes como entradas das chamadas de `split` e `join`. Uma descrição de como funcionam as chamadas já foi oferecida no capítulo de **Implementação** e essas declarações não serão explicadas com relação a sua semântica.

A função seguinte na linha 37 é onde os testes de chamada ao método `Controller::filter_message()` são realizados. Esses testes se caracterizam por instanciar um objeto do tipo `Controller` e usar as variáveis declaradas previamente como entrada para o método em questão. É possível observar 3 chamadas ao método e eles são feitos para testar

1. Uma chamada com parâmetros faltantes
2. Uma chamada correta para realizar a funcionalidade de `split`
3. Uma chamada correta para realizar a funcionalidade de `join`

Dessa forma, todas as funcionalidades principais da aplicação foram testadas cobrindo a maioria de suas facetas e chamadas corretas com algumas verificações a chamadas incorretas. Para uma garantia de que essas funcionalidades não estão gerando vazamentos de memória, podemos realizar a chamada do executável de testes com o uso de um

programa que realiza a verificação de vazamentos de memória e perda de referência a variáveis chamado *valgrind*. Essa chamada pode ser melhor observada na Figura 18

Com relação aos erros existentes, eles fazem referência ao fato de que o atributo interno do objeto `Controller` que encapsula o *Dealer*, `Controller::_dealer` é verificado quando o mesmo é removido da memória. Como temos apenas uma referência para esse atributo através de seu ponteiro, sua verificação gera esse tipo de efeito caso o mesmo não esteja instanciado. Entretanto, é possível que o mesmo tenha sido instanciado pelo usuário final e que ele ainda guarde informações importantes como por exemplo o *número primo* utilizado para construir o Corpo Finito em que o Polinômio foi criado.

Mais a frente é possível observar a declaração de que todos os blocos da *heap* foram liberados e não há possibilidades de vazamentos de memória.

4.2 SEGURANÇA

A respeito da **Análise de Segurança**, podemos comentar tópicos específicos relacionados a proposta feita por Shamir em seu trabalho e relacionar a biblioteca implementada. Entre os pontos aos quais a proposta se manteve fiel ao trabalho proposto por Shamir, podemos citar

1. Uso de um Polinômio para realizar a divisão do Segredo (Dado Original) em partes
2. Uso da Aritmética de Corpos Finitos para a manipulação dos dados
3. Uso de um Polinômio Aleatório único para cada rodada de divisão de Segredo
4. Produção de um número n de partes (*shares*) e da necessidade do uso de um número mínimo t dessas partes para proporcionar a reconstrução do dado original
5. Eventual uso de uma quantidade inferior a t *shares* não torna possível a recuperação de nenhuma informação acerca do dado original
6. Uso de um *número primo* para a geração do Corpo Finito superior ao tamanho de cada byte sendo cifrado

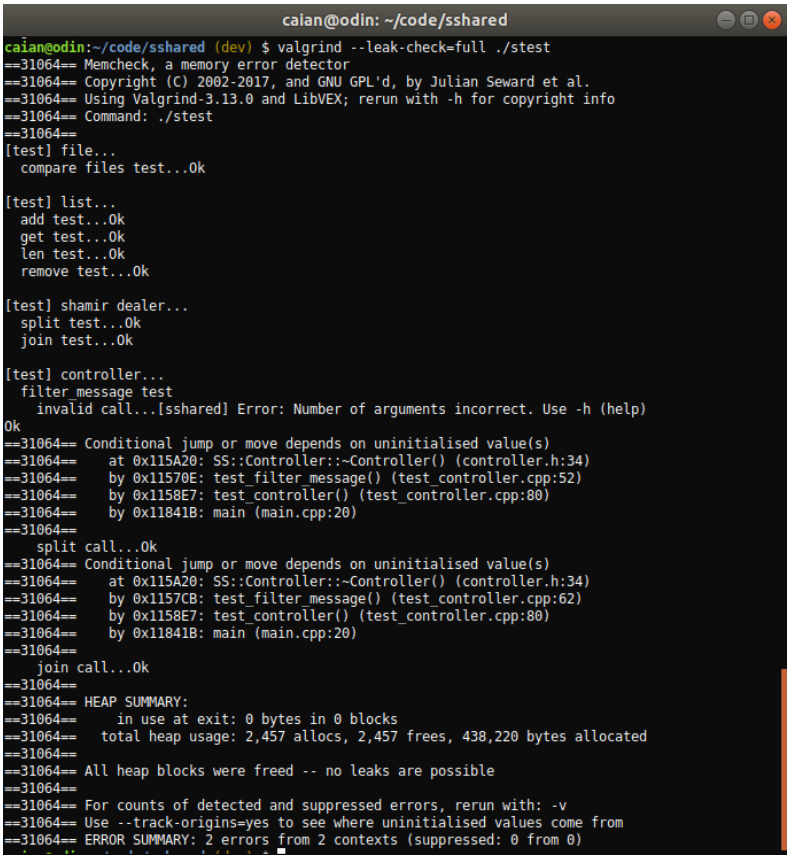

Figura 18 – Execução dos testes com *valgrind*

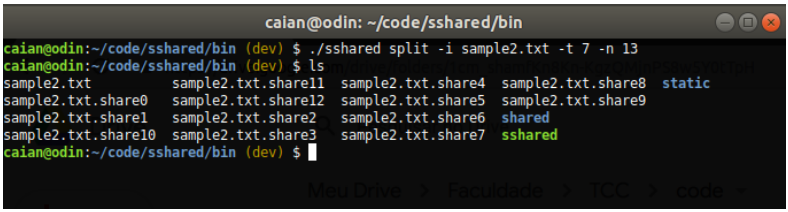

Figura 19 – Criação dos *shares*

Com relação aos primeiros 3 pontos a garantia de Segurança é realizada com o uso da biblioteca matemática **NTL**, uma biblioteca consolidada e seu uso pode ser observado internamente no objeto `ShamirDealer`. Ela garante:

- O uso de operações com números suficientemente grandes para comportar operações com valores acima dos providos pela Linguagem utilizada
- Operações *thread-safe*
- Suporte a Aritmética de Corpos Finitos e de Polinômios dentro de Corpos Finitos
- Suporte a geração de números pseudo-aleatórios dentro do Corpo Finito, que dentro do uso de computadores são o que mais se aproximam da geração de números aleatórios
- Função de reconstrução de polinômios, ou interpolação

Com essas operações internas e encapsuladas por esse objeto as garantias acima desse nível são bem estruturadas. É possível garantir que os outros 3 pontos mencionados possam ser realizados. São os pontos 4, 5 e 6. O ponto 4 já foi observado antes nas capturas de tela de execução e podem ser também observados na Figura 19.

Sobre o ponto 6, é possível usar o número primo padrão da biblioteca ou outro número primo que o usuário final escolha usar em todos os modos de uso da biblioteca. O número primo padrão é muito superior a representação de um byte, que é a unidade máxima de divisão da biblioteca e seu valor é 104471.

Quanto ao ponto 5 também podemos observar a execução de uma chamada a biblioteca pedindo para que ela reconstrua o segredo com uma quantidade menor que a necessária na Figura 20. Nela, vemos

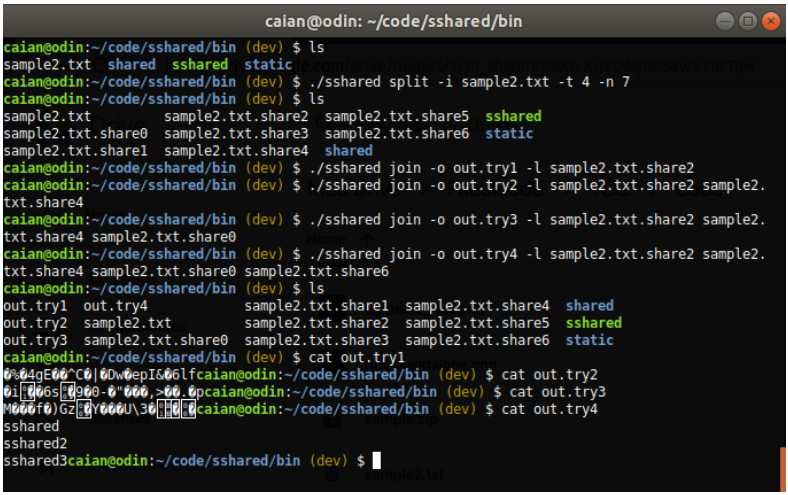

Figura 20 – `Join` com menos *shares* que o necessário

o uso do executável *Standalone* dividir um arquivo em $n = 7$ partes e é necessária uma quantidade mínima de $t = 4$ *shares* para que o arquivo seja reconstruído. Na sequência é possível observar a tentativa de reconstrução do segredo com uma quantidade menor e o resultado dos arquivos pode também ser observado ainda na Figura 20.

Já foi comentado anteriormente sobre uma característica presente na implementação realizada e apresentada nesse trabalho. Quando observamos o uso da biblioteca **sshared**, notamos que a mesma faz a separação do arquivo em *bytes* e para **cada byte** é gerado um polinômio aleatório único dentro do Corpo Finito. O efeito dessa característica é o fato de que nenhuma tupla pode ser utilizada para a reconstrução de um *byte* que não seja o seu *byte* original ainda que os *bytes* sejam idênticos. Essa característica pode ser observada após o uso da biblioteca para realizar o `split` de um arquivo que possui em seu conteúdo apenas *bytes* iguais como mostra a linha a seguir, retirada do arquivo `sample.txt`. O procedimento e os arquivos produzidos são listados na Figura 21 e podemos observar que os valores produzidos são completamente diferentes entre si.

```
1 ssss
```

Agora, realizando uma técnica simples de Criptoanálise e sabendo por exemplo que o arquivo original possui caracteres repetidos

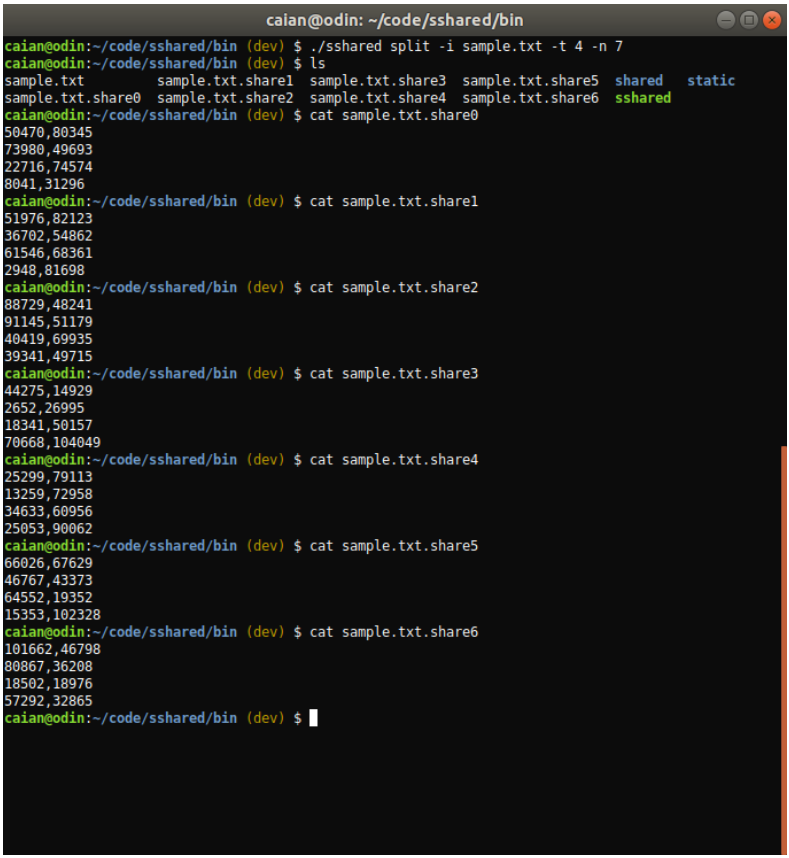

Figura 21 – Operação `Split` e listagem dos *shares*

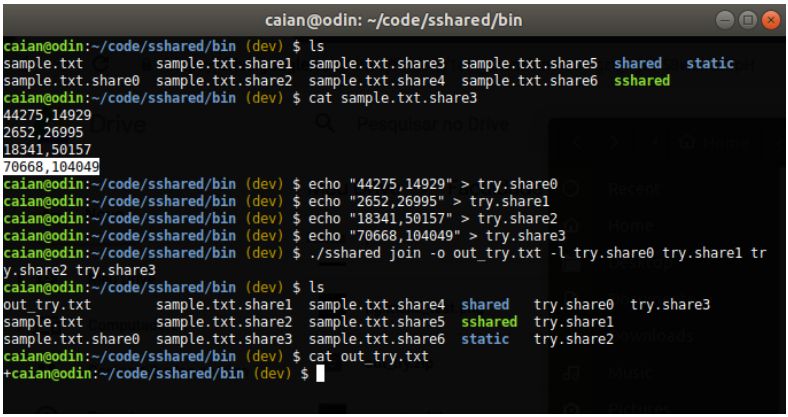

Figura 22 – Exemplo de Ataque 1

mas não tendo qualquer noção de quais caracteres são repetidos nem qual é o caractere e utilizando do fato que temos acesso a um único arquivo de *share* chamado `sample.txt.share3`, podemos realizar um ataque da seguinte forma:

1. Separamos o arquivo único de *share* em 4 arquivos
2. Utilizamos apenas uma linha do arquivo de *share* original em cada um dos 4 arquivos
3. Tentamos a reconstrução do segredo com a funcionalidade `join` da biblioteca

O resultado desse ataque pode ser observado na Figura 22. Podemos observar nela que a eventual reconstrução de um caractere foi possível mas esse caractere não equivale ao caractere original. O arquivo está listado a seguir e pode ser observado na mesma figura Figura 22 em sua última linha.

```
1 +
```

Assumindo ainda que eventualmente nós possamos ter algum tipo de erro e precisemos de mais informações, fomos atrás de outro arquivo de *share* e conseguimos acesso a ele: o arquivo `sample.txt.share5`. Organizamos novamente nosso ataque repetindo o procedimento com o segundo arquivo sendo concatenado ao primeiro em todos os *shares* de tentativa. O procedimento usado para esse ataque será o seguinte:

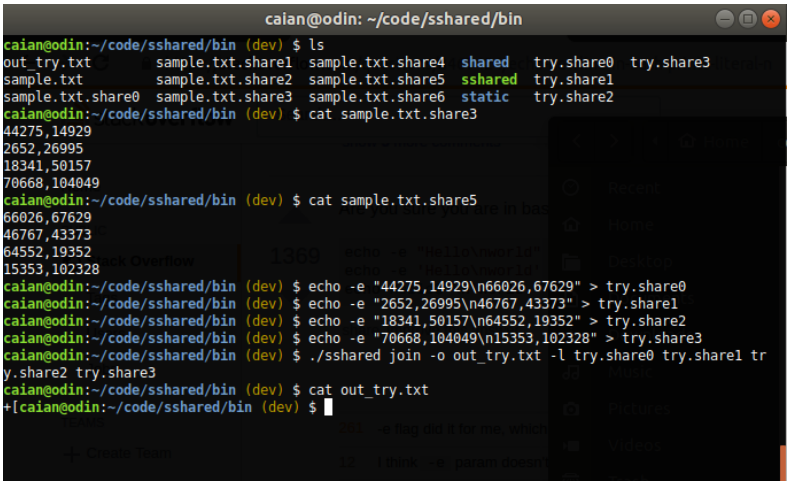

Figura 23 – Exemplo de Ataque 2

1. Concatenamos a **Tupla** do primeiro arquivo de *share* com a **Tupla** respectiva do segundo arquivo de *share* que agora temos acesso

2. Geramos 4 arquivos de *share*, agora com 2 **Tuplas** cada arquivo

3. Tentamos a reconstrução do segredo com a funcionalidade `join` da biblioteca

O resultado desse ataque pode ser observado na Figura 23. Notamos que agora, mesmo tendo recuperado um outro caractere, ele se tornou um caractere diferente do caractere original e também um caractere diferente do primeiro caractere gerado, mostrando que ainda assim esse tipo de ataque não consegue retornar qualquer informação sobre o *Texto em Claro*. O arquivo está listado a seguir também e notamos os caracteres recuperados que diferem entre si e do texto original.

```
1 +]
```

Não obstante com nossa busca em tentar quebrar o segredo, resolvemos tentar uma abordagem diferente de misturar as **Tuplas** de ambos os arquivos entre si. Esse procedimento será o seguinte:

1. Concatenamos a **Tupla** do primeiro arquivo de *share* com a **Tupla** respectiva do segundo arquivo de *share* que temos acesso de

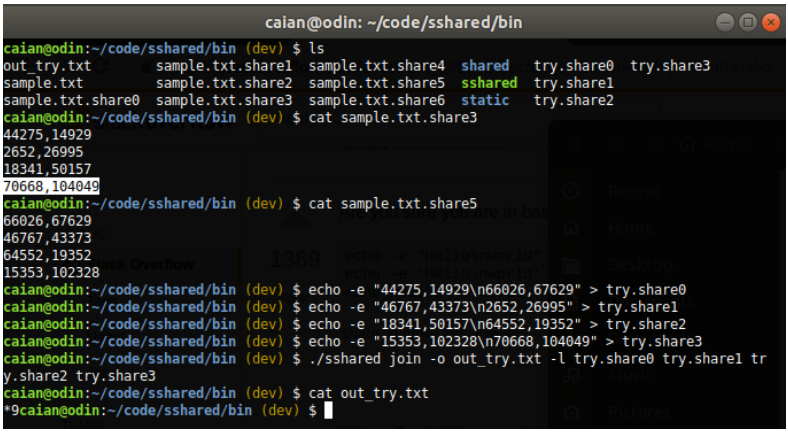

Figura 24 – Exemplo de Ataque 3

maneira alternada

2. Geramos 4 arquivos de *share*, também com 2 **Tuplas** cada arquivo mas de maneira alternada

3. Tentamos a reconstrução do segredo com a funcionalidade `join` da biblioteca

O resultado desse ataque pode ser observado na Figura 24 e o arquivo resultante está mostrado na sequência. Podemos observar que os caracteres recuperados foram completamente dos caracteres recuperados previamente e também diferem dos caracteres contidos no arquivo original.

```
1 *9
```

Podemos concluir então que esse ataque não é efetivo e que se assemelha ao ataque de *Força Bruta*. Entretanto, temos agora acesso a 2 arquivos de *share* e podemos tentar reconstruir o *Texto em Claro* com o uso dos mesmos. Essa reconstrução é relatada na Figura 25 e o arquivo de saída segue na sequência:

```
1 ? \E7\90
```

Podemos notar que os valores recuperados diferem muito dos encontrados previamente com os outros ataques e do arquivo original. Dessa forma, vamos supor que um integrante malicioso nos deu acesso

Figura 25 – Exemplo de Ataque 4

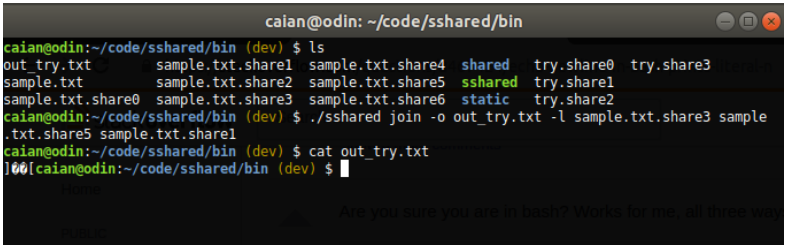

Figura 26 – Exemplo de Ataque 5

a um outro arquivo de *share*: `sample.txt.share1`. Agora podemos tentar mais uma vez reconstruir o Polinômio e tentar novamente recuperar o *Texto em Claro*. Esse procedimento pode ser observado na Figura 26 e o arquivo de saída pode ser visto na sequência:

```
1  ]\A1\AF[
```

Notamos assim que qualquer tentativa de recuperar uma informação referente ao arquivo original com o *Texto em Claro* contido nele é frustrada pela forma que a biblioteca **sshared** foi implementada. Apenas por motivos de completude, será mostrada a recuperação do arquivo original na Figura , onde usamos os arquivos de *shares* citados previamente e um arquivo de *share* a mais, totalizando o mínimo número de arquivos necessários para a reconstrução acontecer.

Podemos facilmente observar que a recuperação do arquivo original foi bem sucedida e o arquivo de saída tem seu conteúdo exatamente igual ao arquivo de entrada da função `split`.

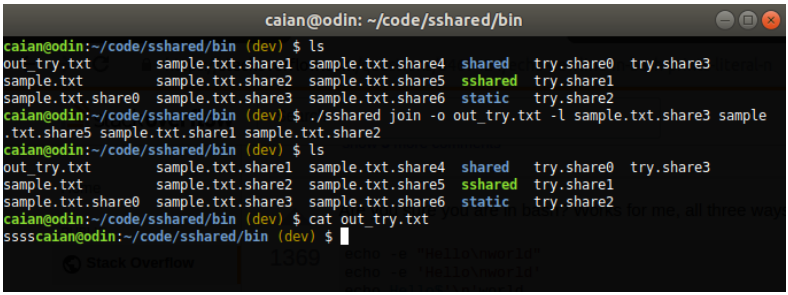

Figura 27 – Recuperação do *Texto em Claro*

5 CONCLUSÃO

Quando observamos o resultado final deste trabalho na forma de uma **Biblioteca de Segredo Compartilhado** de nome **sshared** disponibilizada de maneira aberta e livre é possível dizer que essa carência está sendo suprida com uma proposta válida e possível de utilização, com segurança e possibilidade de escalabilidade, melhorias e acesso fácil aos futuros projetos que podem se originar deste trabalho.

Com uma Arquitetura definida e sempre buscando uma visão de escalabilidade durante seu desenvolvimento, o desenvolvimento futuro de outros Esquemas de Segredo Compartilhado pode ser realizado sem a mudança inicial na Interface de Programação da Aplicação (API) muito similar ao Esquema já implementado de Shamir e que também foi implementado pela **sshared**.

De nome **sshared**, a Biblioteca de Segredo Compartilhado pode ser utilizada em suas 3 formas propostas

- Executável *Standalone*
- Biblioteca de acesso Dinâmico
- Biblioteca de acesso Estático

Bastando apenas ao Programador Usuário ou ao Usuário Final realizar o uso da mesma de uma forma que atenda as suas necessidades. A criação de *scripts* que auxiliam na instalação das dependências necessárias para o correto funcionamento da biblioteca foi realizada e o período de acesso ao uso pode ser encurtado com a execução desses arquivos. Todas as configurações necessárias para o primeiro uso da biblioteca são quase que inexistentes após a execução desses *scripts*. O uso da **sshared** também é de fácil compreensão ao usuário final visto que suas operações são descritivas (`split` e `join`) se assemelhando muito a outras aplicações de mesmo caráter existentes como alternativas ofrecidas por outras implementações.

Em questão a Análise de Segurança e de Modelagem, funcionalidades como a Geração de Valores Aleatórios é feita com uma relativa melhor qualidade em comparação a algumas outras alternativas. Também existe o suporte a números de precisão maior do que as oferecidas pela linguagem disponibiliza através do uso da biblioteca matemática **NTL** que traz consigo ainda operações *thread-safe*, suporte a Aritmética de Corpos Finitos e geração de Polinômios dentro desses Corpos.

Como forma a garantir ainda mais Segurança ao Programador que faz uso da biblioteca, ele pode optar por realizar uma quantidade maior de configurações internas que apenas a **sshared** oferece em comparação com as outras alternativas:

- Possibilidade de mudar o número primo que gera o Corpo Finito base para a geração do Polinômio
- Possibilidade de geração de um Polinômio de tamanho maior ao tamanho oferecido por outras alternativas

Como benefício adicional que podemos citar, a **sshared** realiza a geração de um polinômio único para cada rodada em que ela executa o procedimento de `split` que é feito para cada *byte* do arquivo a ser cifrado, garantindo uma segurança ao Usuário Final de que caso algum atacante consiga partes suficientes de *shares* para a reconstrução de uma pequena quantidade de *bytes* do *Texto em Claro* essas partes dos *shares* obtidos não poderão ser reutilizadas para a recuperação do resto do *Texto em Claro*. Isso garante que ataques mencionados por Blakley em seu trabalho como *Incidentes de Abnegação*, *Traição* ou *Combinação* possam ter sua eficácia reduzida na recuperação do *Texto em Claro*. O Atacante precisaria ter em sua posse a **totalidade** dos arquivos de *share* necessários para a reconstrução do *Texto em Claro*.

A implementação da **sshared** traz consigo também uma rodada de análises e testes que possibilitam a garantia ao Usuário Final de que essas falhas e brechas já estão validadas e não são aberturas a um eventual atacante. Também dá ao mesmo indivíduo a possibilidade de seguir por si os mesmos testes e realizar essas verificações iniciais e outras que poderiam ser formas de ataques que venham a calhar no futuro com o auxílio de uma base de ideias ofertadas de maneira descritiva.

Verificações internas ao código foram feitas através de ferramentas já consolidadas como *Valgrind* e *gdb* e dão ainda mais embasamento aos testes produzidos que podem ser encontrados junto ao código e compilados em forma de um executável de testes e validados. Esses resultados estão também disponíveis no trabalho em questão e podem ser também avaliados ou acrescidos em caso de trabalhos futuros.

5.1 TRABALHOS FUTUROS

Por fim mas não menos importante, quando se comenta sobre o término de um trabalho como este é comum que se chegue a conclusão

de que este encerramento não é um fim, mas um início de uma ferramenta que possui um vasto campo para melhorias. A implementação e o uso de outras tecnologias podem trazer ainda mais qualidade a este trabalho mas não é o único campo de estudo que pode ser abordado.

Como forma a enumerar as abordagens futuras, foi elaborada uma lista com alguns pontos de crescimento:

- Implementação de uma maior cobertura de testes
- Melhorias a respeito do padrão de arquivo de saída utilizado no trabalho e de nome `.share`
- Suporte a versões mais atuais da biblioteca **NTL** e de suas dependências internas que podem vir a trazer inconsistências
- Maior cobertura de testes e verificações acerca de possíveis ataques como *side-channel attack* em suas diversas formas
- Implementação de outros **Esquemas de Segredo Compartilhado**

Como o último tema aborda especificamente um tópico relacionado ao tema de Segredo Compartilhado, alguns outros Esquemas sugeridos para estudo e possível implementação são:

- *Verifiable Secret Sharing Scheme*, trabalho de Pedersen (PEDERSEN, 1991)
- *Non-iteractive Verifiable Secret Sharing Scheme*, trabalho de Feldman (FELDMAN, 1987)
- *Proactive Secret Sharing*, trabalho de Herzberg, Jarecki, Krawczyk e Yung (**??**)
- Esquemas baseados em Quadrados Latinos, como os sugeridos pelo trabalho de Cooper, Donovan e Seberry (COOPER; DONOVAN; SEBERRY, 1994)

ANEXO A – Artigo

Implementação e Análise de uma Biblioteca de Segredo Compartilhado

Ilê Caian Gums[1]

[1]Universidade Federal de Santa Catarina (UFSC)

caian.gums@grad.ufsc.br

Abstract. *This paper describes the develop of the library* ***sshared****, a library of Secret Sharing open source and free that was produced as Undergraduate Thesis in Federal University of Santa Catarina. It was developed with the language C++, offers the implementation of Secret Sharing with the Scheme described by Shamir and known as Threshold Scheme or (k. n) threshold scheme and allows extensions for others Schemes as future works.*

Resumo. *Este artigo descreve a criação da biblioteca* ***sshared****, uma biblioteca de Segredo Compartilhado livre e aberta que foi produzida como Trabalho de Conclusão de Curso da Universidade Federal de Santa Catarina. Ela foi desenvolvida em linguagem C++, oferece a implementação do Esquema de Segredo Compartilhado descrito por Shamir de nome Esquema de Limiar ou (k, n) threshold scheme e permite a extensão para outros Esquemas como trabalhos futuros.*

1. Introdução

A necessidade de proteção a dados confidenciais sempre foi uma constante em diversos momentos da história. O conhecimento acerca de conceitos como Criptografia, Segurança e Sigilo dos Dados e Comunicação Segura são temas recorrentes em vários ambientes diferentes apenas do Estado da Arte em segurança em Computação. Este trabalho implementa uma alternativa ao uso compartilhado de algum recurso de maneira segura em forma de uma biblioteca de Segredo Compartilhado na linguagem C++.

Questões de Segurança e Implementação serão discutidas após a apresentação da definição de Segredo Compartilhado e uma breve Análise será feita sobre alguns possíveis ataques. Por fim serão apresentadas as Considerações Finais e os Trabalhos Futuros.

2. Segredo Compartilhado

O conceito deriva da busca da solução de um problema proposto por Liu [Liu 1968]. O problema descrevia que existiam 11 cientistas trabalhando em um projeto secreto e que havia a necessidade de guardar o projeto em um armário que pudesse ser aberto apenas na presença de 6 deles ou mais e com a condição de que com 5 ou menos cientistas presentes o armário não pudesse ser aberto.

Shamir [Shamir 1979] nos oferece uma solução para o problema e discute a inviabilidade do uso de Chaves Criptográficas na resolução do problema e outros

problemas com escalabilidade do mesmo. Na sequência ele propõe um esquema que ficou conhecido como **Esquema de Limiar** ou *(k, n) Threshold Scheme*. Shamir definiu o esquema da seguinte maneira

Definição 1. Esquema de Limiar: Queremos compartilhar um dado D em n partes $D_1, D_2, \ldots, D_3$ de forma que as seguintes propriedades sejam respeitadas:

1) com o conhecimento de k ou mais partes é facilmente recuperável o dado original D
2) com o conhecimento de $k - 1$ ou menos partes não é possível recuperar qualquer informação sobre o dado original D

Entre seu trabalho, Shamir comenta que a utilização desse Esquema seria possível com o auxílio de um Polinômio $q(x)$ e que bastaria escolher um Polinômio aleatório de grau k e avaliar o Polinômio em n valores aleatórios respeitando $n > 0$ e utilizar o segredo D como termo independente do Polinômio.

Para a recuperação do Polinômio basta reunir as partes e recuperar o mesmo através de Interpolação Polinomial e avaliar o Polinômio com o valor $x = 0$ e o resultado seria o segredo recuperado $q(0) = D$.

3. Proposta

A biblioteca **sshared** foi implementada na linguagem C++ e oferece o uso em formato de **executável** bem como arquivos para inclusão **estática** e **dinâmica**. Ela oferece duas funcionalidades básicas: *split* e *join*.

A funcionalidade de *split* realiza a separação de um arquivo em um número n de partes e com a necessidade da presença de k partes para a recuperação do arquivo original. Já a funcionalidade *join* nada sabe sobre a quantidade k mínima necessária para a recuperação do segredo e sempre tem como saída um arquivo. Caso a quantidade k de arquivos mínima seja passada o arquivo original é recuperado. Caso não seja possível recuperar o arquivo, é escrito um arquivo com dados que nada fazem referência ao arquivo original.

A biblioteca possui uma API de acesso que vai de encontro a outras bibliotecas avaliadas no processo de desenvolvimento e possui a funcionalidade de mudança do número *primo* que irá gerar o Corpo no qual serão produzidos os Polinômios aleatórios, uma funcionalidade não identificada em outras implementações conhecidas.

É válido citar que em vez de utilizar apenas um único Polinômio aleatório para todo o arquivo ou para cada byte do arquivo a biblioteca realiza a criação de um Polinômio aleatório para cada byte e este é avaliado com valores aleatórios distintos entre cada polinômio.

O formato de saída de arquivos carrega a tupla necessária para a reconstrução de cada byte do arquivo e tem a extensão de nome *.share* com o número sequencial de produção que nada tem a ver com o valor do byte, o Polinômio ou o valor avaliado no Polinômio. Ele possui em seu interior a tupla de avaliação *(x, q(x))* necessária para a reconstrução de cada Polinômio.

Foram também realizados testes de funcionalidades básicas durante o desenvolvimento como o uso da *Lista* implementada, da abstração que trata de *Arquivos*(criação, abertura e leitura). As funcionalidades do *Controlador* e do *Dealer* implementado que levou o nome do criador do seu Esquema, *ShamirDealer* também foram feitas e todos esses testes podem ser observados através de um executável de testes.

As funções matemáticas foram realizadas através da biblioteca matemática **NTL** que possui suporte a valores numéricos superiores aos ofertados pela linguagem C++ e operações sobre Corpos Finitos e geração de valores aleatórios. Para a instalação da **NTL** e de suas dependências, foi produzido um script de instalação que realizada a correta instalação da mesma.

4. Análise

Para verificações de execução e vazamento de memória foram realizados testes com o auxílio da ferramenta *Valgrind* e execuções de alguns cenários possíveis de teste. A ferramenta *Valgrind* alertou para que não existem vazamentos de memória durante a execução dos testes de todas as funcionalidades básicas da biblioteca e garantem ao programador que fizer uso da mesma essa garantia.

Quanto aos cenários de avaliações, os testes ocorreram através da tentativa de reconstrução de um arquivo com menor quantidade de partes necessárias. Também foi testada a recuperação de um arquivo com uma quantidade de partes menor do que a necessária mas com o uso dos arquivos conhecidos para construção de um número mínimo de partes necessárias tomando ciência de que o arquivo original possuía apenas um único byte repetido diversas vezes. Em ambos os casos não foi possível realizar a recuperação do arquivo original e apenas com a quantidade mínima de arquivos necessários foi possível realizar a recuperação do mesmo.

Não foram realizados testes com relação a ataques do tipo *side-channel*, que avaliam questões como uso de energia ou uso de recursos pela CPU sem diretamente ter acesso aos dados que estão sendo trafegados ou cifrados.

5. Conclusões

Como resultado em forma de uma ferramenta que pode ser utilizada de maneira livre e aberta, este trabalho disponibiliza aos que querem fazer o uso da funcionalidade de Segredo Compartilhado uma alternativa válida e possível de utilização.

Aos usuários, é oferecida uma biblioteca que realiza as funções básicas com um controle maior do que as outras ferramentas existentes. A ferramenta pode ser acessada de maneiras diferentes e o porte da mesma para outras linguagens é facilitado pela escolha da linguagem. Também existem testes que garantem um grau de segurança contra os casos mostrados e abrem ao desenvolvedor futuro uma maior possibilidade de acréscimo destes testes. Testes relacionados aos tipos de ataques *side-channel* são deixadas aos futuros desenvolvedores que podem realizar esse tipo de implementação.

A possibilidade de escalabilidade é oferecida para os futuros desenvolvedores através da implementação de outros *Dealers* e respeitando a API interna de acesso que

não difere de outras ferramentas existentes. Aos que desejam realizar implementações de outros Esquemas, são sugestões:

- *Verifiable Secret Sharing Scheme*, trabalho de Pedersen [Pedersen 1991]
- *Proactive Secret Sharing*, trabalho de Herzberg, Jarecki, Krawczyk e Yung [Herzberg 1995]
- Esquemas baseados em Quadrados Latinos, sugeridos por Cooper, Donovan e Seberry [Cooper 1994]

6. Referências

Cooper, J.; Donovan, D.; Seberry, J. (1994) Secret sharing schemes arising from latin squares.

Herzberg, A., Jarecki, S., Krawczyk, H. and Yung, M (1995) Proactive secret sharing or: How to cope with perpetual leakage. In: SPRINGER. Annual International Cryptology Conference. [S.l.]. p. 339–352.

Liu, C. L. (1968) Introduction to combinatorial mathematics. McGraw-Hill.

Pedersen, T. P. (1991) Non-interactive and information-theoretic secure verifiable secret sharing. In: SPRINGER.Annual International Cryptology Conference. [S.l.]. p. 129–140.

Shamir, A. (1979) How to share a secret. Communications of the ACM, AC m, v. 22, n. 11, p. 612–613.

ANEXO B – Código

B.1.1 README.md

````
1  # sshared
2
3  A C++ implementation of Secret Sharing.
4
5  Supported Implementations:
6  - Shamir
7
8  # Development Requirements
9
10 To development you must use a Linux (Debian/Ubuntu
       if possible). The internal requirements to
       develop can be installed with the `ntl-install.
       sh` script inside the `scripts/` directory.
11
12 # Test
13
14 The following commands should create and run the
       test program:
15
16 ```bash
17 $ make test
18 $ ./stest
19 ```
20
21 # Compile
22
23 If you want to compile and use, the lib can be
       compile and used in 3 forms:
24
25 * Standalone
26 ```bash
27 make
28 ```
29 * Dynamic
30 ```bash
31 make dynamic
````

```
32 ```
33 * Static
34 ```bash
35 make static
36 ```
37
38 This will generate the files inside the `bin/`
       directory and you can use them as you wish.
39
40 # Usage
41
42 If you don't know how to use the lib you can try to
       compile the `Standalone` targed and execute
43 ```bash
44 ./bin/sshared -h
45 ```
46
47 Or see the test files for how you can use and
       create your application using this lib!
```

B.1.2 makefile

```
1  CXX=g++
2  CFLAGS= -std=c++11
3  DYNAMIC_FLAGS=-fPIC
4  DEBUG=-g
5  CFLAGS_DEBUG= -Wall $(DEBUG) $(CFLAGS)
6
7  AR=ar
8  ARFLAGS=-rcs
9
10 INCLUDES=-I./include -I$(HOME)/sw/include
11 LIBS=-L$(HOME)/sw/lib -lntl -lgmp -lgf2x -lm
12
13
14 # this is to avoid files, directories and targets
       with the same name
15 # let 'make' assume that is all up to date
16 .PHONY: all test clean
17
18 # source files
```

```
19 SRC= src/controller.cpp \
20          src/shamir_dealer.cpp \
21          util/file.cpp \
22          util/file_handler.cpp
23
24 # source files
25 STATIC_OBJ_PATH= bin/static/
26 SHARED_OBJ_PATH= bin/shared/
27
28 # standalone main
29 STANDALONE_MAIN= src/main_standalone.cpp
30
31 # test source files
32 TST_SRC= src/controller.cpp \
33          src/shamir_dealer.cpp \
34          util/file.cpp \
35          util/file_handler.cpp \
36          test/test_controller.cpp \
37          test/test_shamir_dealer.cpp \
38          test/test_file.cpp \
39          test/test_list.cpp \
40          test/main.cpp
41
42 # output
43 OUT=bin/sshared
44
45 # static lib name
46 OUT_STATIC=bin/sshared.a
47
48 # shared lib name
49 OUT_SHARED=bin/sshared.so
50
51 # test output
52 TST_OUT=stest
53
54 # test shares output
55 TST_SHARES=*.share*
56
57 # compile all
58 all:
59          $(CXX) $(CFLAGS) $(INCLUDES) -o $(OUT) $(
```

```
            SRC) $(STANDALONE_MAIN) $(LIBS)
60
61  # compile static
62  static:
63          $(CXX) $(CFLAGS) $(INCLUDES) -c src/
                controller.cpp -o $(STATIC_OBJ_PATH)
                controller.o
64          $(CXX) $(CFLAGS) $(INCLUDES) -c src/
                shamir_dealer.cpp -o $(STATIC_OBJ_PATH)
                shamir_dealer.o
65          $(CXX) $(CFLAGS) $(INCLUDES) -c util/file.
                cpp -o $(STATIC_OBJ_PATH)file.o
66          $(CXX) $(CFLAGS) $(INCLUDES) -c util/
                file_handler.cpp -o $(STATIC_OBJ_PATH)
                file_handler.o
67          $(AR) $(ARFLAGS) $(OUT_STATIC) $(
                STATIC_OBJ_PATH)*
68
69  # compile dynamic
70  dynamic:
71          $(CXX) $(DYNAMIC_FLAGS) $(CFLAGS) $(
                INCLUDES) -c src/controller.cpp -o $(
                SHARED_OBJ_PATH)controller.o
72          $(CXX) $(DYNAMIC_FLAGS) $(CFLAGS) $(
                INCLUDES) -c src/shamir_dealer.cpp -o $
                (SHARED_OBJ_PATH)shamir_dealer.o
73          $(CXX) $(DYNAMIC_FLAGS) $(CFLAGS) $(
                INCLUDES) -c util/file.cpp -o $(
                SHARED_OBJ_PATH)file.o
74          $(CXX) $(DYNAMIC_FLAGS) $(CFLAGS) $(
                INCLUDES) -c util/file_handler.cpp -o $
                (SHARED_OBJ_PATH)file_handler.o
75          $(CXX) -shared $(CFLAGS) $(SHARED_OBJ_PATH)
                * -o $(OUT_SHARED)
76
77  # compile test
78  test:
79          $(CXX) $(CFLAGS_DEBUG) $(INCLUDES) -o $(
                TST_OUT) $(TST_SRC) $(LIBS)
80
81  clean:
```

```
82          rm $(OUT) $(TST_OUT) $(OUT_STATIC) $(
                OUT_SHARED)
83
84  veryclean:
85          rm $(OUT) $(STATIC_OBJ_PATH)* $(
                SHARED_OBJ_PATH)* $(OUT_STATIC) $(
                OUT_SHARED) $(TST_OUT) $(TST_SHARES)
```

B.2 DIRETÓRIO INCLUDE

B.2.1 controller.h

```
1  #ifndef CONTROLLER_H
2  #define CONTROLLER_H
3
4  #include <string>
5
6  // Util
7  #include "list.h"
8
9  // Dealers
10 #include "shamir_dealer.h"
11
12 // File Handler
13 #include "file_handler.h"
14
15 // Used in error log
16 #include <iostream>
17
18 namespace SS
19 {
20     // List of Evaluated Shares as TupleLists
21     typedef List<TupleList*> EvaluatedShares;
22
23     class Controller {
24     public:
25
26         Controller() {
27             this->_n = 0;
28             this->_t = 0;
29             this->_p = 104471;
```

```
30              this->_dealer_type = "shamir";
31          }
32
33          ~Controller() {
34              if(!!this->_list_file_path) delete this
                    ->_list_file_path;
35          }
36
37          /* filter_message filter the recieved
                message from input
38          * program
39          *
40          *  @param  mes[]       list of parameters
41          *  @param  size        list size
42          *
43          *  @return true on success or false on fail
44          */
45          bool filter_message(const char* mes[], int
                size);
46
47      private:
48          unsigned long _n;
49          unsigned long _t;
50          unsigned long _p;
51          std::string _file_path;
52          std::string _out_file_path;
53          StringList* _list_file_path;
54          // Dealer type
55          std::string _dealer_type;
56          // Dealer Object
57          Dealer* _dealer;
58          // EvaluatedShares of Shares
59          EvaluatedShares* _es;
60
61
62          /* set_value is a general setting value to
                filer_message.
63          *
64          *  @param  art         argument name
65          *  @param  value       argument value
66          *
```

```
67          *  @return true if all goes ok, false on
                error.
68          */
69          bool set_value(const char* arg, const char*
                value);
70
71          // inside set's
72          void set_t(const char* value);
73          void set_n(const char* value);
74          void set_p(const char* value);
75          void set_file_path(const char* value);
76          void set_out_file_path(const char* value);
77          void set_dealer_type(const char* value);
78
79          bool set_list_filepath(const char** arg,
                int index, int size);
80
81          // print help information
82          void print_help();
83
84          /** Split a content as std::string
85           *
86           *  @return void
87           */
88          void split();
89
90          /** Join a content as std::string
91           *
92           *  @return void
93           */
94          void join();
95
96      };
97  }
98  #endif
```

B.2.2 dealer.h

```
1  #ifndef DEALER_H
2  #define DEALER_H
3
```

```
4  #include <string>
5  #include "list.h"
6  #include "tuple.h"
7
8  // Used in error log
9  #include <iostream>
10
11 namespace SS
12 {
13     // Value x of q(x)
14     typedef std::string ssX;
15     // Value of q(x) based on x
16     typedef std::string ssY;
17
18     // Tuple definition
19     typedef Tuple<ssX, ssY> ShareTuple;
20
21     // List of shares
22     typedef List<ShareTuple> TupleList;
23
24     class Dealer {
25     public:
26
27         virtual ~Dealer() { }
28
29         /* [VIRTUAL] split method splits a message
               or data in 'n' parts with a
30         * minimum 't' parts.
31         *
32         *  @param  t            minimum number os
               parts required to
33         *                       reconstruct the
               secret
34         *  @param  n            number of parts to
               be splited
35         *
36         *  @return TupleList object with all the
               parts or
37         *  NULL on fail
38         */
39         virtual TupleList* split(std::string data,
```

```
            unsigned int t, unsigned int n) = 0;
40
41          /* [VIRTUAL] join method try reconstruct
                the secret with the given parts
42          *
43          *  @param  shares      parts of the secret
                as a TupleList
44          *
45          *  @return std::string of the join
                operation or NULL on fail
46          */
47          virtual std::string join(TupleList* shares)
                = 0;
48
49      };
50  }
51
52  #endif
```

B.2.3 file_handler.h

```
1   #ifndef FILEHANDLER_H
2   #define FILEHANDLER_H
3
4   #include <string>
5   #include <sstream>
6
7   #include "list.h"
8   #include "readable_file.h"
9   #include "writable_file.h"
10
11  namespace SS
12  {
13      typedef List<std::string> StringList;
14
15      class FileHandler {
16      public:
17          /** Split a string based on a delimiter and
                returns a StringList*
18           *
19           *  @param  data        std::string to be
```

```
             splited
20           *  @param  delimiter   character to be
                 used as delimiter on split
21           *
22           *  @return StringList* with splited std::
                 strings
23           */
24          static StringList* split_string(std::string
                 data, char delimiter);
25
26          /** Write a content inside a file
27           *
28           *  @param  content     std::string to be
                 written
29           *  @param  to          filepath of the
                 file
30           *
31           *  @return void
32           */
33          static void write(std::string content, std
                ::string to);
34
35          /** Read a file and place the content
                inside a std::string
36           *
37           *  @param  from        filepath of the
                 file
38           *
39           *  @return std::string with the content of
                  the file
40           */
41          static std::string read(std::string from);
42      };
43  }
44
45  #endif
```

B.2.4 file.h

```
1  #ifndef FILE_H
2  #define FILE_H
```

```
3
4  #include <iostream>
5  #include <fstream>
6
7  namespace SS
8  {
9      class File {
10     public:
11
12         File(std::string path) : fp(path) { }
13         ~File() { }
14
15         /* open method open a file with the given
               path on object creation
16         *
17         *  @return void.
18         */
19         void open();
20
21         /* close method closes the file that was
               opened
22         *
23         *  @return void.
24         */
25         void close();
26
27     protected:
28         // file path
29         std::string fp;
30         // file stream
31         std::fstream fs;
32
33     };
34 }
35
36 #endif
```

B.2.5 list.h

```
1  #ifndef LIST_H
2  #define LIST_H
```

```
3
4  #include <vector>
5
6  namespace SS
7  {
8      template <class T>
9      class List {
10     public:
11
12         // Constructor
13         List();
14         ~List();
15
16         /* add method adds the given element el on
               the
17         * list
18         *
19         *  @param  el      element to be inserted
20         *
21         *  @return void
22         */
23         void add(T el);
24
25         /* remove method remove the last inserted
               element
26         *
27         *  @return void
28         */
29         void remove();
30
31         /* len method return the size of the list
32         *
33         *  @return the size of the list
34         */
35         unsigned int len();
36
37         /* get method get the reference to the
               given element
38         * on specific position
39         *
40         *  @param  pos     position of the element
```

```
41              on
42          *                       the list
43          *
44          *   @return the element or null
45          */
46          T get(unsigned int pos);
47
48      private:
49          // attributes
50          // inside list
51          std::vector<T>* list;
52
53      };
54  }
55
56  template <class T>
57  SS::List<T>::List() {
58      this->list = new std::vector<T>();
59  }
60
61  template <class T>
62  SS::List<T>::~List() {
63      this->list->clear();
64      delete this->list;
65  }
66
67  template <class T>
68  void SS::List<T>::add(T el) {
69      this->list->push_back(el);
70  }
71
72  template <class T>
73  void SS::List<T>::remove() {
74      this->list->pop_back();
75  }
76
77  template <class T>
78  T SS::List<T>::get(unsigned int pos) {
79      return this->list->at(pos);
80  }
```

```
81 template <class T>
82 unsigned int SS::List<T>::len() {
83     return this->list->size();
84 }
85
86 #endif
```

B.2.6 readable_file.h

```
1  #ifndef READABLEFILE_H
2  #define READABLEFILE_H
3
4  #include "file.h"
5
6  namespace SS
7  {
8      class ReadableFile : public File {
9      public:
10
11         ReadableFile(std::string path) : File(path)
               { }
12
13         /* read method reads a file with the given
               path on object creation
14         *
15         *  @return std::string with file data.
16         */
17         std::string read();
18
19     };
20 }
21
22 #endif
```

B.2.7 writable_file.h

```
1  #ifndef WRITABLEFILE_H
2  #define WRITABLEFILE_H
3
4  #include "file.h"
5
```

```
 6 namespace SS
 7 {
 8     class WritableFile : public File {
 9     public:
10
11         WritableFile(std::string path) : File(path)
               { }
12
13         /* write method writes a file with the
               given path on object creation
14         *
15         *  @param  val          value to be written
16         *
17         *  @return void
18         */
19         void write(std::string val);
20
21     private:
22
23         /* file_exist method check if a file exist.
                If not, create a file.
24         * This method is useful on write files
25         *
26         *  @return true if file exists or false if
               doesn't.
27         */
28         bool file_exist();
29
30         /* file_create method create the file if
               does not exist
31         * on disk
32         *
33         *  @return void
34         */
35         void file_create();
36
37     };
38 }
39
40 #endif
```

B.2.8 shamir_dealer.h

```
1  #ifndef SHAMIRDEALER_H
2  #define SHAMIRDEALER_H
3
4  #include <NTL/ZZ_p.h>
5  #include <NTL/ZZ_pX.h>
6  #include "dealer.h"
7
8  namespace SS
9  {
10     class ShamirDealer : public Dealer {
11     public:
12
13         ShamirDealer(unsigned long prime) : p(prime
               ) { }
14
15         /* split method splits a message or data in
                'n' parts with a
16         * minimum 't' parts.
17         *
18         *  @param  t            minimum number os
               parts required to
19         *                       reconstruct the
               secret
20         *  @param  n            number of parts to
               be splited
21         *
22         *  @return TupleList object with all the
               parts or
23         *  NULL on fail
24         */
25         TupleList* split(std::string data, unsigned
                int t, unsigned int n);
26
27         /* join method try reconstruct the secret
               with the given parts
28         *
29         *  @param  shares       parts of the secret
               as a TupleList
```

```
30          *
31          *  @return std::string of the join
                operation or NULL on fail
32          */
33          std::string join(TupleList* shares);
34
35      private:
36          unsigned long p;
37      };
38  }
39
40  #endif
```

B.2.9 tuple.h

```
1   #ifndef TUPLE_H
2   #define TUPLE_H
3
4   #include <iostream>
5
6   namespace SS
7   {
8       template<class S, class R>
9       class Tuple {
10      private:
11          S _first;
12          R _second;
13      public:
14          Tuple(S first, R second): _first(first),
                _second(second) {}
15          ~Tuple() {}
16
17          // Get of parameters
18          S first() { return _first; }
19          R second() { return _second; }
20
21          friend std::ostream & operator<<(std::
                ostream & os, Tuple t) {
22              os << t.first() << "," << t.second();
23              return os;
24          }
```

```
25      };
26  }
27
28  #endif
```

B.3 DIRETÓRIO SCRIPTS

B.3.1 ntl-install.sh

```
1  #
       ==========================================================
        #
2  #
       #
3  #                    NTL-install script
                            #
4  #
       #
5  #    This script install the NTL, GMP and gf2x libs
       to use   #
6  # in sshared project.
                                                  #
7  #    This script works for 64bits UNIX-like systems.
        For     #
8  # more instructions, access the official NTL docs.
                 #
9  #
       #
10 #    ERRORS: See the error section
                                    #
11 #
       #
12 #
       ==========================================================
        #
13
14 # INSTALL DIR
```

```
15 INSTALL_DIR='ntl_install'
16 # VERSIONS
17 NTL='ntl-10.3.0'
18 GMP='gmp-6.1.2'
19 GF2X='gf2x-1.1'
20
21 # Setup
22 mkdir ${INSTALL_DIR}
23 cd ${INSTALL_DIR}
24
25 # < DOWNLOAD >
26 echo 'Downloading files...'
27 # NTL
28 wget http://www.shoup.net/ntl/${NTL}.tar.gz
29 # GMP
30 wget https://gmplib.org/download/gmp/${GMP}.tar.bz2
31 # gf2x
32 wget http://gforge.inria.fr/frs/download.php/file
       /36934/${GF2X}.tar.gz
33 echo 'Download done!'
34 # < \DOWNLOAD >
35
36 # < EXTRACT >
37 echo 'Extract files...'
38 # NTL
39 tar -xvzf ${NTL}.tar.gz
40 # GMP
41 tar -jxvf ${GMP}.tar.bz2
42 # gf2x
43 tar -xvzf ${GF2X}.tar.gz
44 echo 'Extract done!'
45 # < \EXTRACT >
46
47 # < INSTALL >
48 echo 'Install section...'
49 # < GMP >
50     echo 'Installing GMP'
51     cd ${GMP}
52     ./configure --prefix=$HOME/sw
53     make
54     make check
```

```
55      make install
56      # return to root directory
57      cd ..
58      echo 'Installing GMP done!'
59  # < \GMP >
60  # < GF2X >
61      echo 'Installing gf2x'
62      cd ${GF2X}
63      ./configure --prefix=$HOME/sw ABI=64 CFLAGS='-
            m64 -O2'
64      make
65      make check
66      make install
67      echo 'Symbolic links creation'
68      sudo ln -s $HOME/sw/lib/libgf2x.so.1.0.0 /usr/
            lib/libgf2x.so.1.0.0
69      sudo ln -s $HOME/sw/lib/libgf2x.so.1.0.0 /usr/
            lib/libgf2x.so.1
70      sudo ln -s $HOME/sw/lib/libgf2x.so.1.0.0 /usr/
            lib/libgf2x.so
71      # return to root directory
72      cd ..
73      echo 'Installing gf2x done!'
74  # < \GF2x >
75  # < NTL >
76      echo 'Installing NTL (this may take some time)'
77      cd ${NTL}/src
78      ./configure PREFIX=$HOME/sw NTL_GF2X_LIB=on
            GF2X_PREFIX=$HOME/sw GMP_PREFIX=$HOME/sw
79      make
80      make check
81      make install
82      # return to root directory
83      cd ../..
84      echo 'Installing NTL done!'
85  # < \GF2x >
86
87  # Cleanup
88  cd ..
89  rm -rf ${INSTALL_DIR}
90
```

```
91 echo 'Install done! :)'
92 # < \INSTALL >
93
94 #
    ================================================================
     #
95 #

    #
96 #                    POSSIBLE ERRORS
                              #
97 #

    #
98 #   If you find any errors on install, try run
    these        #
99 # commands:

    #
100 #   sudo apt-get install m4
                                        #
101 #   sudo apt-get install g++
                                      #
102 #

    #
103 #
    ================================================================
     #
```

B.4 DIRETÓRIO SRC

B.4.1 controller.cpp

```
1 #include "controller.h"
2
3 bool SS::Controller::filter_message(const char* mes
    [], int size) {
4
5     // initial check
6     if(size < 2) {
```

```
7           std::cerr << "[sshared] Error: Number of
                arguments incorrect. Use -h (help)" <<
                std::endl;
8           return false;
9       }
10
11      // print help
12      if(mes[1][1] == 'h') {
13          this->print_help();
14          return true;
15      }
16
17      // check if split or join was passed
18      std::string sj = mes[1];
19      if(sj.compare("split") != 0 && sj.compare("join
            ") != 0) {
20          std::cerr << "[sshared] Error: Pass the
                operation (split or join). Use -h (help
                )" << std::endl;
21          return false;
22      }
23      int nsize = size - 1;
24
25      // check for number of arguments
26      if(nsize < 2 || (sj.compare("join") > 0 &&
            nsize % 2 == 0)) {
27          std::cerr << "[sshared] Error:  Number of
                arguments incorrect. Use -h (help)" <<
                std::endl;
28          return false;
29      }
30
31      // set values on Controller
32      for(int i = 2; i < nsize; i += 2) {
33          if(mes[i][1] == 'l') {
34              if(!this->set_list_filepath(mes, i,
                    size - i)) {
35                  std::cerr << "[sshared] Error:
                        Number of arguments incorrect.
                        Use -h (help)" << std::endl;
36                  return false;
```

```
37                  }
38                  break;
39              }
40              if(!this->set_value(mes[i], mes[i+1])) {
41                  return false;
42              }
43          }
44
45          // do split/join operation
46          if(sj.compare("split") == 0) {
47              this->split();
48          } else if (sj.compare("join") == 0) {
49              this->join();
50          } else {
51              return false;
52          }
53
54          return true;
55
56      }
57
58      void SS::Controller::split() {
59          if(this->_dealer_type.empty()) {
60              std::cerr << "[sshared] Error: No Dealer
                    Type defined" << std::endl;
61          }
62          if(this->_dealer_type.compare("shamir") == 0) {
63              // read file
64              std::string data = SS::FileHandler::read(
                    this->_file_path);
65
66              this->_dealer = new ShamirDealer(this->_p);
67              this->_es = new EvaluatedShares();
68              SS::TupleList* tl;
69
70              // step 1 - split each byte
71              for(unsigned int i = 0; i < data.length() -
                     1; i++) {
72                  tl = this->_dealer->split(data.substr(i
                        , 1), this->_t, this->_n);
73                  this->_es->add(tl);
```

```
74          }
75
76          // step 2 - save to files
77          for(unsigned int i = 0; i < this->_n; i++)
                {
78              std::string out_share = "";
79              // write the Shares on each file
80              for(unsigned int j = 0; j < this->_es->
                    len(); j++) {
81                  tl = this->_es->get(j);
82                  SS::ShareTuple st = tl->get(i);
83                  out_share += st.first() + "," + st.
                        second() + "\n";
84              }
85
86              std::string share_file_name = this->
                    _file_path + ".share" + std::
                    to_string(i);
87              SS::FileHandler::write(out_share,
                    share_file_name);
88          }
89
90          // cleanup
91          for (unsigned int i = 0; i < this->_es->len
                (); i++) {
92              delete this->_es->get(i);
93          }
94          delete this->_es;
95          delete this->_dealer;
96          return;
97      }
98      std::cerr << "[sshared] Error: Dealer Type not
            supported" << std::endl;
99  }
100
101 void SS::Controller::join() {
102     if(this->_dealer_type.empty()) {
103         std::cerr << "[sshared] Error: No Dealer
                Type defined" << std::endl;
104     }
105     if(this->_dealer_type.compare("shamir") == 0) {
```

```
106
107             this->_es = new EvaluatedShares();
108
109             // step 1 - open files and get the shares
110             for(unsigned int i = 0; i < this->
                    _list_file_path->len(); i++) {
111                 // read the shares
112                 std::string data = SS::FileHandler::
                        read(this->_list_file_path->get(i))
                        ;
113
114                 SS::StringList* lstring = SS::
                        FileHandler::split_string(data, '\n
                        ');
115
116                 for(unsigned int j = 0; j < lstring->
                        len(); j++) {
117                     SS::TupleList* tl;
118                     // check for tuples on es
119                     if(this->_es->len() == 0 || j >=
                            this->_es->len()) {
120                         tl = new SS::TupleList();
121                         this->_es->add(tl);
122                     }
123                     // build the tuples
124                     std::string line = lstring->get(j);
125                     SS::StringList* elements = SS::
                            FileHandler::split_string(line,
                             ',');
126                     SS::ssX x = elements->get(0);
127                     SS::ssY y = elements->get(1);
128                     SS::ShareTuple st(x, y);
129                     tl = this->_es->get(j);
130                     tl->add(st);
131                     delete elements;
132                 }
133
134                 delete lstring;
135             }
136
137             // step 2 - join the shares
```

```
138             this->_dealer = new ShamirDealer(this->_p);
139             std::string out = "";
140 
141             for(unsigned int i = 0; i < this->_es->len
                    (); i++) {
142                 SS::TupleList* tl = this->_es->get(i);
143                 out += this->_dealer->join(tl);
144             }
145 
146             SS::FileHandler::write(out, this->
                    _out_file_path);
147 
148             // cleanup
149             for (unsigned int i = 0; i < this->_es->len
                    (); i++) {
150                 delete this->_es->get(i);
151             }
152             delete this->_dealer;
153             delete this->_es;
154             return;
155         }
156         std::cerr << "[sshared] Error: Dealer Type not
                supported" << std::endl;
157     }
158 
159     bool SS::Controller::set_value(const char* arg,
            const char* value) {
160 
161         if(arg[0] != '-') {
162             std::cerr << "[ssshared] Error: Invalid
                    argument passed: " << arg << std::endl;
163             return false;
164         }
165 
166         switch(arg[1]) {
167             // input file path
168             case 'i': {
169                 this->set_file_path(value);
170                 break;
171             }
172             // output file path
```

```
173             case 'o': {
174                 this->set_out_file_path(value);
175                 break;
176             }
177             // n value
178             case 'n': {
179                 this->set_n(value);
180                 break;
181             }
182             // t value
183             case 't': {
184                 this->set_t(value);
185                 break;
186             }
187             // p value
188             case 'p': {
189                 this->set_p(value);
190                 break;
191             }
192             // _dealer type value
193             case 'd': {
194                 this->set_dealer_type(value);
195                 break;
196             }
197             default:
198                 std::cerr << "[sshared] Error: Invalid
                        option: " << arg << std::endl;
199                 return false;
200                 break;
201         }
202         return true;
203     }
204
205     void SS::Controller::set_t(const char* value) {
206         std::string v(value);
207         try {
208             this->_t = std::stoul(v);
209         } catch (const std::invalid_argument& ia) {
210              std::cerr << "[sshared] Error: Invalid
                    argument on t definition" << std::endl
                    ;
```

```
211         }
212     }
213
214     void SS::Controller::set_n(const char* value) {
215         std::string v(value);
216         try {
217             this->_n = std::stoul(v);
218         } catch (const std::invalid_argument& ia) {
219             std::cerr << "[sshared] Error: Invalid
                argument on n definition" << std::endl
                ;
220         }
221     }
222
223     void SS::Controller::set_p(const char* value) {
224         std::string v(value);
225         try {
226             this->_p = std::stoul(v);
227         } catch (const std::invalid_argument& ia) {
228             std::cerr << "[sshared] Error: Invalid
                argument on p definition" << std::endl
                ;
229         }
230     }
231
232     void SS::Controller::set_file_path(const char*
        value) {
233         this->_file_path = value;
234     }
235
236     void SS::Controller::set_out_file_path(const char*
        value) {
237         this->_out_file_path = value;
238     }
239
240     bool SS::Controller::set_list_filepath(const char**
         value, int index, int size) {
241         this->_list_file_path = new StringList();
242         for(int i = index+1; i < index+size; i++) {
243             if (value[i][0] == '-') {
244                 delete this->_list_file_path;
```

```
245                 return false;
246             }
247             std::string file_path = value[i];
248             this->_list_file_path->add(file_path);
249         }
250         return true;
251     }
252
253     void SS::Controller::set_dealer_type(const char*
        value) {
254         this->_dealer_type = value;
255     }
256
257     void SS::Controller::print_help() {
258         std::cout << "sshared lib help information: "
            << std::endl;
259         std::cout << "Usage:" << std::endl;
260         std::cout << "./sshared (<operation> <args>)|(-
            h)" << std::endl;
261         std::cout << "  operation: the operation that
            will be performed" << std::endl;
262         std::cout << "      - split" << std::endl;
263         std::cout << "      - join" << std::endl;
264         std::cout << "  arguments: each argument must
            be followed by respective values" << std::
            endl;
265         std::cout << "      - i: input file(for split)"
             << std::endl;
266         std::cout << "      - l: list of input files(in
             case of join) -- must be the LAST argument
             with values" << std::endl;
267         std::cout << "      - o: output file name(in
            case of join)" << std::endl;
268         std::cout << "      - t: minimum shares" << std
            ::endl;
269         std::cout << "      - n: number of shares" <<
            std::endl;
270         std::cout << "      - d: dealer type(shamir
            default)" << std::endl;
271         std::cout << "      - p: prime number used
            (104471 default)" << std::endl;
```

```
272     std::cout << "      - h: help information" <<
            std::endl;
273 }
```

B.4.2 main_standalone.cpp

```
 1 #include <iostream>
 2 #include <string>
 3
 4 #include "controller.h"
 5 #include "shamir_dealer.h"
 6
 7 #include "readable_file.h"
 8 #include "writable_file.h"
 9
10
11 int main(int argc, const char* argv[]) {
12
13     SS::Controller* con = new SS::Controller();
14     con->filter_message(argv, argc);
15
16     // cleanup
17     delete con;
18     return 0;
19 }
```

B.4.3 shamir_dealer.cpp

```
 1 #include "shamir_dealer.h"
 2
 3 SS::TupleList* SS::ShamirDealer::split(std::string
       data, unsigned int t, unsigned int n) {
 4
 5     // Error check
 6     if(t == 0) {
 7         std::cerr << "[sshared] Error: t value
               invalid" << std::endl;
 8         return NULL;
 9     }
10     if(n == 0) {
```

```
11          std::cerr << "[sshared] Error: n value
                invalid" << std::endl;
12          return NULL;
13      }
14      if(n < t) {
15          std::cerr << "[sshared] Error: n value
                invalid(n<t)" << std::endl;
16          return NULL;
17      }
18
19      SS::TupleList* rv = new SS::TupleList();
20
21      // Init ZZ_p
22      NTL::ZZ prime = NTL::conv<NTL::ZZ>(this->p);
23      NTL::ZZ_p::init(prime);
24
25      NTL::ZZ_pX pol;
26
27      const char * data_char = data.c_str();
28      unsigned long d = static_cast<unsigned long>(
            data_char[0]);
29
30      for(unsigned long i = 1; i < t; i++) {
31          NTL::ZZ_p coef = NTL::random_ZZ_p();
32          NTL::SetCoeff(pol, i, coef);
33      }
34      NTL::SetCoeff(pol, 0, d);
35
36      for(unsigned int i = 0; i < n; i++) {
37          NTL::ZZ_p val = NTL::random_ZZ_p();
38          SS::ssY eval = std::to_string(NTL::conv<
                long>(NTL::eval(pol, val)));
39          SS::ssX index = std::to_string(NTL::conv<
                long>(val));
40          SS::ShareTuple tuple(index, eval);
41
42          rv->add(tuple);
43      }
44
45      return rv;
46  }
```

```
47
48 std::string SS::ShamirDealer::join(SS::TupleList*
       shares) {
49
50     // Error check
51     if(!shares) {
52         std::cerr << "[sshared] Error: no shares
               passed" << std::endl;
53         return NULL;
54     }
55
56     // Init ZZ_p
57     NTL::ZZ prime = NTL::conv<NTL::ZZ>(this->p);
58     NTL::ZZ_p::init(prime);
59
60     // create the vectors of coefficients and index
61     NTL::Vec<NTL::ZZ_p> coef;
62     NTL::Vec<NTL::ZZ_p> ind;
63
64     // sets length of vectors
65     coef.SetLength((long) shares->len());
66     ind.SetLength((long) shares->len());
67
68     for(long i = 0; i < shares->len(); i++) {
69         ind[i] = NTL::conv<NTL::ZZ_p>(std::stoul(
               shares->get(i).first()));
70         coef[i] = NTL::conv<NTL::ZZ_p>(std::stoul(
               shares->get(i).second()));
71     }
72
73     // recreate the polynomial
74     NTL::ZZ_pX pol;
75
76     // interpolate to find the polynomial
77     pol = NTL::interpolate(ind, coef);
78
79     NTL::ZZ_p zero_index;
80     zero_index = NTL::conv<NTL::ZZ_p>((long) 0);
81     int result = NTL::conv<int>(NTL::eval(pol,
           zero_index));
82     char result_char = static_cast<char>(result);
```

```
83      std::string eval(1, result_char);
84
85      return eval;
86  }
```

B.5 DIRETÓRIO TEST

B.5.1 main_test.h

```
1  #ifndef MAIN_TEST_H
2  #define MAIN_TEST_H
3
4  #include <iostream>
5  #include <string>
6
7  // Controller include
8  #include "controller.h"
9
10 // Dealer includes
11 #include "shamir_dealer.h"
12
13 // File includes
14 #include "readable_file.h"
15 #include "writable_file.h"
16
17 // List include
18 #include "list.h"
19
20 /* controller tests */
21 void test_controller();
22
23 /* list tests */
24 void test_list();
25
26 /* file tests */
27 void test_file();
28
29 /* shamir_dealer tests */
30 void test_shamir_dealer();
31
32 #endif
```

B.5.2 main.cpp

```
1  /* main tests include */
2  #include "main_test.h"
3
4  // main test function
5  int main(int argc, char* argv[]) {
6
7      std::cout << "[test] file... " << std::endl;
8      test_file();
9
10     std::cout << std::endl;
11     std::cout << "[test] list... " << std::endl;
12     test_list();
13
14     std::cout << std::endl;
15     std::cout << "[test] shamir dealer... " << std
           ::endl;
16     test_shamir_dealer();
17
18     std::cout << std::endl;
19     std::cout <<"[test] controller... " << std::
           endl;
20     test_controller();
21
22     return 0;
23 }
```

B.5.3 test_controller.cpp

```
1  /* test header include */
2  #include "main_test.h"
3
4  static const int argc_error = 1;
5  static const char* argv_error[argc_error] = {""};
6
7  static const int argc_split = 10;
8  static const char* argv_split[argc_split] = {
9      "sshare",
10     "split",
```

```
11      "-i",
12      "sample2.txt",
13      "-t",
14      "3",
15      "-n",
16      "5",
17      "-p",
18      "104471"
19  };
20
21  static const int argc_join = 12;
22  static const char* argv_join[argc_join] = {
23      "sshare",
24      "join",
25      "-p",
26      "104471",
27      "-o",
28      "out_sample.txt",
29      "-l",
30      "sample2.txt.share0",
31      "sample2.txt.share1",
32      "sample2.txt.share2",
33      "sample2.txt.share3",
34      "sample2.txt.share4"
35  };
36
37  void test_filter_message() {
38
39      std::cout << "  filter_message test";
40
41      SS::Controller* con = new SS::Controller();
42      bool check = true;
43
44      std::cout << std::endl;
45      std::cout << "    invalid call...";
46      check = con->filter_message(argv_error,
            argc_error);
47      if (check) {
48          std::cout << "Error" << std::endl;
49      } else {
50          std::cout << "Ok" << std::endl;
```

```
51      }
52      delete con;
53      con = new SS::Controller();
54
55      std::cout << "    split call...";
56      check = con->filter_message(argv_split,
            argc_split);
57      if(!check) {
58          std::cout << "Error" << std::endl;
59      } else {
60          std::cout << "Ok" << std::endl;
61      }
62      delete con;
63      con = new SS::Controller();
64
65      std::cout << "    join call...";
66      check = con->filter_message(argv_join,
            argc_join);
67      if(!check) {
68          std::cout << "Error" << std::endl;
69      } else {
70          std::cout << "Ok" << std::endl;
71      }
72
73      // cleanup
74      delete con;
75  }
76
77  /* controller tests */
78  void test_controller() {
79
80      test_filter_message();
81
82  }
```

B.5.4 test_file.cpp

```
1  /* main tests include */
2  #include "main_test.h"
3
4  /* file tests */
```

```
5  void test_file() {
6
7      // file name
8      std::string fp = "test.txt";
9
10     // writable file
11     SS::WritableFile* wf = new SS::WritableFile(fp)
           ;
12     wf->open();
13     std::string wfbuf = "Test file.\n1 2 3 test";
14     wf->write(wfbuf);
15     wf->close();
16     delete wf;
17
18     // readable file
19     SS::ReadableFile* rf = new SS::ReadableFile(fp)
           ;
20     rf->open();
21     std::string rfbuf = rf->read();
22     rf->close();
23     delete rf;
24
25     bool check = true;
26     // TEST: compare test
27     std::cout << "  compare files test...";
28     if(rfbuf.compare(wfbuf) == 0) check = true;
29     else check = false;
30     if(!check) {
31         std::cout << "\nError on compare" << std::
               endl;
32         return;
33     }
34
35     std::cout << "Ok" << std::endl;
36
37 }
```

B.5.5 test_list.cpp

```
1  /* main tests include */
2  #include "main_test.h"
```

```
3
4  /* list tests */
5  void test_list() {
6      // SS::List creation
7      SS::List<int>* il = new SS::List<int>();
8      SS::List<std::string>* sl = new SS::List<std::
           string>();
9
10     // TEST: add test
11     bool check = true;
12     std::cout << "  add test...";
13
14     // int test
15     int el0 = 3;
16     int el1 = 5;
17     int el2 = 7;
18     il->add(el0);
19     il->add(el1);
20     il->add(el2);
21     if(il->get(0) != el0) check = false;
22     if(il->get(1) != el1) check = false;
23     if(il->get(2) != el2) check = false;
24
25     if(!check) {
26         std::cout << "Error on int" << std::endl;
27         delete il;
28         delete sl;
29         return;
30     }
31
32     // string test
33     std::string se0 = "test0";
34     std::string se1 = "test2";
35     std::string se2 = "testtest";
36     sl->add(se0);
37     sl->add(se1);
38     sl->add(se2);
39     if(sl->get(0).compare(se0) != 0) check = false;
40     if(sl->get(1).compare(se1) != 0) check = false;
41     if(sl->get(2).compare(se2) != 0) check = false;
42
```

```
43      if(!check) {
44          std::cout << "Error on string" << std::endl
                ;
45          delete il;
46          delete sl;
47          return;
48      }
49
50      std::cout << "Ok" << std::endl;
51
52      // TEST: get test
53      std::cout << "  get test...";
54
55      // int test
56      if(il->get(0) != el0) check = false;
57      if(il->get(1) != el1) check = false;
58      if(il->get(2) != el2) check = false;
59
60      if(!check) {
61          std::cout << "Error on int" << std::endl;
62          delete il;
63          delete sl;
64          return;
65      }
66
67      // string test
68      if(sl->get(0).compare(se0) != 0) check = false;
69      if(sl->get(1).compare(se1) != 0) check = false;
70      if(sl->get(2).compare(se2) != 0) check = false;
71      if(!check) {
72          std::cout << "Error on string" << std::endl
                ;
73          delete il;
74          delete sl;
75          return;
76      }
77
78      std::cout << "Ok" << std::endl;
79
80      // TEST: len test
81      std::cout << "  len test...";
```

```
82
83      // int test
84      if(il->len() != 3) check = false;
85      if(!check) {
86          std::cout << "Error on int" << std::endl;
87          delete il;
88          delete sl;
89          return;
90      }
91
92      // string test
93      if(sl->len() != 3) check = false;
94      if(!check) {
95          std::cout << "Error on string" << std::endl
                ;
96          delete il;
97          delete sl;
98          return;
99      }
100
101     std::cout << "Ok" << std::endl;
102
103     // TEST: remove test
104     std::cout << "  remove test...";
105
106     // int test
107     il->remove();
108     il->remove();
109     il->remove();
110     if(il->len() != 0) check = false;
111     if(!check) {
112         std::cout << "Error on int" << std::endl;
113         delete il;
114         delete sl;
115         return;
116     }
117
118     // string test
119     sl->remove();
120     sl->remove();
121     sl->remove();
```

```
122     if(sl->len() != 0) check = false;
123     if(!check) {
124         std::cout << "Error on string" << std::endl
                ;
125         delete il;
126         delete sl;
127         return;
128     }
129
130     std::cout << "Ok" << std::endl;
131
132     // cleanup
133     delete il;
134     delete sl;
135
136     return;
137 }
```

B.5.6 test_shamir_dealer.cpp

```
 1 /* main tests include */
 2 #include "main_test.h"
 3
 4 // Mock values
 5 static const unsigned long p = 104471;
 6 static const std::string data = "a";
 7
 8 // static const ShareTuple share_0("0", "97");
 9 static const SS::ShareTuple share_1("1", "111");
10 static const SS::ShareTuple share_2("2", "187");
11 static const SS::ShareTuple share_3("3", "379");
12 static const SS::ShareTuple share_4("4", "741");
13
14 void test_split_shamir_dealer() {
15
16     std::cout << "  split test...";
17
18     SS::ShamirDealer* sd = new SS::ShamirDealer(p);
19     bool check = true;
20
21     SS::TupleList* tl;
```

```
22      unsigned int t = 3;
23      unsigned int n = 5;
24      tl = sd->split(data, t, n);
25
26      if(!tl) check = false;
27      if(tl->len() != n) check = false;
28
29      // cleanup
30      if(!!tl) delete tl;
31      if(!!sd) delete sd;
32
33      if(!check) {
34          std::cout << "Error" << std::endl;
35          return;
36      }
37
38      std::cout << "Ok" << std::endl;
39  }
40
41  void test_join_shamir_dealer() {
42
43      std::cout << "  join test...";
44
45      SS::ShamirDealer* sd = new SS::ShamirDealer(p);
46      bool check = true;
47
48      SS::TupleList* tl = new SS::TupleList();
49      // Match the known values from split
50      // the secret must not be known
51      // tl->add(share_0); // 'a' = 97
52      tl->add(share_1);
53      tl->add(share_2);
54      tl->add(share_3);
55      tl->add(share_4);
56      std::string rv = sd->join(tl);
57
58      if(rv.empty()) check = false;
59      if(rv.compare(data) != 0) check = false;
60
61      // cleanup
62      if(!!tl) delete tl;
```

```
63      if(!!sd) delete sd;
64
65      if(!check) {
66          std::cout << "Error" << std::endl;
67          return;
68      }
69
70      std::cout << "Ok" << std::endl;
71  }
72
73  /* shamir_dealer tests */
74  void test_shamir_dealer() {
75
76      test_split_shamir_dealer();
77      test_join_shamir_dealer();
78
79  }
```

B.6 DIRETÓRIO UTIL

B.6.1 file_handler.cpp

```
1  #include "file_handler.h"
2
3
4  std::string SS::FileHandler::read(std::string from)
       {
5      SS::ReadableFile* in_file = new SS::
           ReadableFile(from);
6      in_file->open();
7      std::string data = in_file->read();
8      in_file->close();
9      delete in_file;
10     return data;
11 }
12
13 void SS::FileHandler::write(std::string content,
       std::string to) {
14     SS::WritableFile* out_file = new SS::
           WritableFile(to);
15     out_file->open();
```

```
16      out_file->write(content);
17      out_file->close();
18      delete out_file;
19  }
20
21  SS::StringList* SS::FileHandler::split_string(std::
        string data, char delimiter) {
22      std::stringstream ssdata(data);
23      std::string token;
24      SS::StringList* lstring = new SS::StringList();
25      while(std::getline(ssdata, token, delimiter)) {
26          lstring->add(token);
27      }
28      return lstring;
29  }
```

B.6.2 file.cpp

```
1  #include <string>
2  #include "readable_file.h"
3  #include "writable_file.h"
4
5  void SS::File::open() {
6      if(this->fp.empty()) {
7          // Error: No filepath
8          std::cerr << "[File] No filepath\n";
9          return;
10     }
11     if(this->fs.is_open()) {
12         // Error: File already opened
13         std::cerr << "[File] File already opened\n"
               ;
14         return;
15     }
16
17     this->fs.open(this->fp.c_str());
18 }
19
20 void SS::File::close() {
21     if(!this->fs.is_open()) {
22         // No file opened
```

```
23          std::cerr << "[File] No open file\n";
24          return;
25      }
26
27      this->fs.close();
28  }
29
30  std::string SS::ReadableFile::read() {
31      if(!this->fs.is_open()) {
32          // No file opened
33          std::cerr << "[File] No open file\n";
34          return NULL;
35      }
36
37      std::string buf = "";
38      std::string line;
39      unsigned int i = 0;
40      while(!this->fs.eof()) {
41          std::getline(this->fs, line);
42          buf += line;
43          if(!this->fs.eof()) buf += "\n";
44          i++;
45      }
46      return buf;
47
48  }
49
50  void SS::WritableFile::write(std::string val) {
51      if(!this->fs.is_open()) {
52          if(this->fp.empty()) {
53              // No file path
54              std::cerr << "[File] No file path\n";
55              return;
56          }
57          if(!this->file_exist()) {
58              this->file_create();
59          }
60          this->open();
61      }
62
63      this->fs << val;
```

```
64 }
65
66 bool SS::WritableFile::file_exist() {
67     if (FILE *f = fopen(this->fp.c_str(), "r")) {
68         fclose(f);
69         return true;
70     } else {
71         return false;
72     }
73 }
74
75 void SS::WritableFile::file_create() {
76     std::ofstream os (this->fp);
77     os.close();
78 }
```

REFERÊNCIAS

ABRAHAM, O.; SHEFIU, G. O. An improved caesar cipher (icc) algorithm. *International Journal Of Engineering Science & Advanced Technology (IJESAT)*, v. 2, p. 1198–1202, 2012.

AKI, S. G. Digital signatures: a tutorial survey. *Computer*, IEEE, n. 2, p. 15–24, 1983.

ATKINS, D.; STALLINGS, W.; ZIMMERMANN, P. *PGP Message Exchange Formats*. [S.l.], August 1996. `http://www.rfc-editor.org/rfc/rfc1991.txt`. <http://www.rfc-editor.org/rfc/rfc1991.txt>.

BIHAM, E.; SHAMIR, A. Differential cryptanalysis of des-like cryptosystems. *Journal of Cryptology*, v. 4, n. 1, p. 3–72, 1991. ISSN 1432-1378. <`https://doi.org/10.1007/BF00630563`>.

BLAKLEY, G. R. et al. Safeguarding cryptographic keys. In: *Proceedings of the national computer conference*. [S.l.: s.n.], 1979. v. 48, p. 313–317.

CALLAS, J. et al. *OpenPGP Message Format*. [S.l.], November 1998. `http://www.rfc-editor.org/rfc/rfc2440.txt`. <http://www.rfc-editor.org/rfc/rfc2440.txt>.

CALLAS, J. et al. *OpenPGP Message Format*. [S.l.], November 2007. `http://www.rfc-editor.org/rfc/rfc4880.txt`. <http://www.rfc-editor.org/rfc/rfc4880.txt>.

COOPER, J.; DONOVAN, D.; SEBERRY, J. Secret sharing schemes arising from latin squares. 1994.

FELDMAN, P. A practical scheme for non-interactive verifiable secret sharing. In: IEEE. *Foundations of Computer Science, 1987., 28th Annual Symposium on*. [S.l.], 1987. p. 427–438.

FIPS, P. 186-2. digital signature standard (dss). *Federal Information Processing Standards Publication*, v. 2, 2000.

FIPS, P. 186-3: Digital signature standard (dss). *Federal Information Processing Standards Publication*, v. 3, 2009.

GOYAL, K.; KINGER, S. Modified caesar cipher for better security enhancement. *International Journal of Computer Applications*, Foundation of Computer Science, v. 73, n. 3, 2013.

KALISKI, B. *PKCS #1: RSA Encryption Version 1.5*. [S.l.], March 1998. `http://www.rfc-editor.org/rfc/rfc2313.txt`. <http://www.rfc-editor.org/rfc/rfc2313.txt>.

LAMB, C. *pythongfshare*. [S.l.]: Chris Lamb, 2017. `https://chris-lamb.co.uk/posts/python-gfshare-secret-sharing-in-python`.

LIU, C. L. Introduction to combinatorial mathematics. McGraw-Hill, 1968.

MEHURON, W. Digital signature standard (dss). us department of commerce, national institute of standards and technology (nist). *Information Technology Laboratory (ITL). FIPS PEB*, v. 186, 1994.

MORIARTY, K. et al. *PKCS #1: RSA Cryptography Specifications Version 2.2*. [S.l.], November 2016.

PEDERSEN, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. In: SPRINGER. *Annual International Cryptology Conference*. [S.l.], 1991. p. 129–140.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, ACM, v. 21, n. 2, p. 120–126, 1978.

SHAMIR, A. How to share a secret. *Communications of the ACM*, ACm, v. 22, n. 11, p. 612–613, 1979.

SHIREY, R. *Internet Security Glossary*. [S.l.], May 2000.

SILVERSTONE, D.; MCVITTIE, S. *libgfshare*. [S.l.]: Gitano, 2006. `https://git.gitano.org.uk/libgfshare.git/`.

SPRENKELS, D. *randombytes*. [S.l.]: Daan Sprenkels, 2016. `https://github.com/dsprenkels/randombytes/`.

SPRENKELS, D. *sss lib*. [S.l.]: Daan Sprenkels, 2017. `https://github.com/dsprenkels/sss/`.

STALLINGS, W. *Cryptography and Network Security: Principles and Practice*. [S.l.]: Pearson Upper Saddle River, NJ, 2005.

TANENBAUM, A. S. *Modern operating system.* [S.l.]: Pearson Education, Inc, 2009.