Universidade Federal de Pernambuco - UFPE
Centro de Informática - CIn
Bacharelado em Ciência da Computação

Projeto de Conclusão de Disciplina

Merkle Trees em Python

Recife, Outubro de 2022

UFPE

Universidade Federal de Pernambuco - UFPE
Centro de Informática - CIn

Bacharelado em Ciência da Computação

Disciplina: Criptografia
Professor: Ruy Queiroz
Período Letivo: 2022.1
Aluna: Amanda Nunes Silvestre Costa

Recife, Outubro de 2022

Sumário

1. Contexto do Projeto

Este projeto foi realizado pela aluna Amanda Costa para a disciplina Criptografia, ministrada pelo professor Ruy Queiroz.

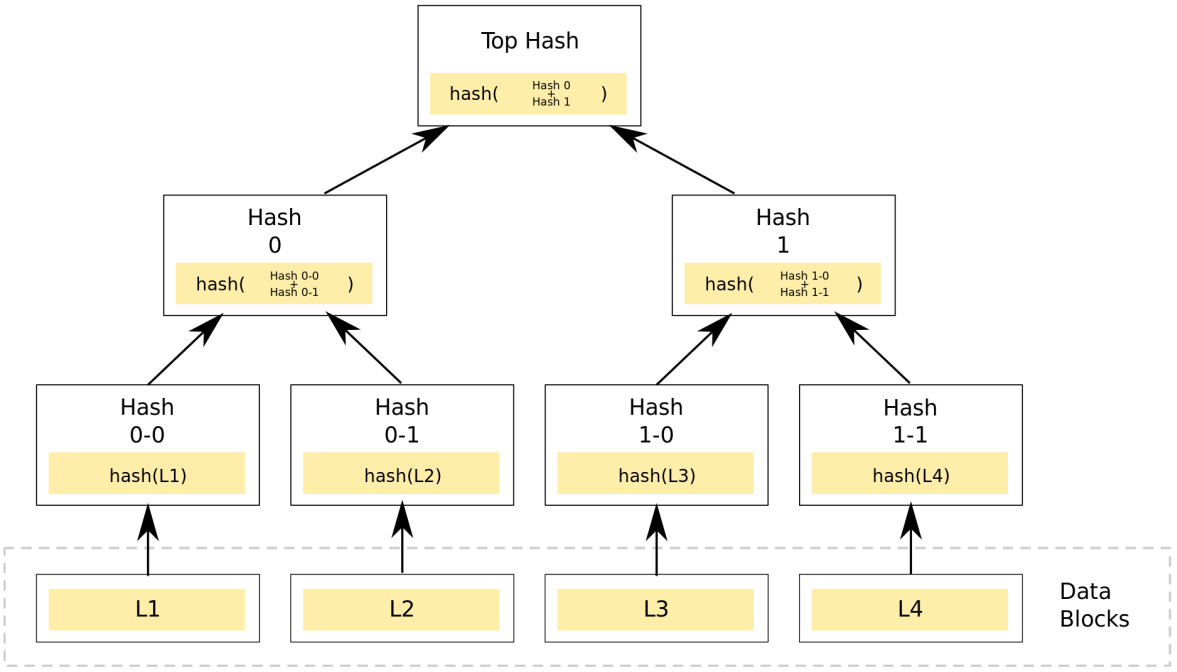
O objetivo do projeto é, além de entender os conceitos relacionados à Merkle Trees e sua importância dentro da criptografia moderna, construir parte do código necessário para geração de uma Merkle Proof.

Tiramos aqui um tempo para a explicação dos conceitos base do projeto, a parte de implementação pode ser encontrada no próximo tópico.

1.1 Merkle Tree

1) Estrutura

Merkle Tree é uma estrutura de dados baseada em hash, sua representação é uma árvore onde cada folha contém o hash criptográfico de um bloco de dados e todos os nós além das folhas (aqui conhecidos como “nós internos”) contém o hash de combinação de seus nós filhos.



Representação de Merkle Tree

Como mostrado na figura, Merkle Trees normalmente são implementadas com árvores binárias (como é o caso do nosso programa), mas podem ser criadas com qualquer árvore n-ária.

2) Merkle Proof

Para qualquer prova usando uma Merkle Tree, devemos ter antes salvo a Merkle Root, raiz da árvore gerada com os blocos de dados, na imagem acima representada como Top Hash.

Utilizaremos aqui o exemplo de um upload de arquivo num sistemas de arquivos online, como o Dropbox, e posterior download do mesmo arquivo, mas a mesma lógica é aplicada para qualquer prova.

Suponha que tenhamos, ao enviar um grupo de arquivos e sua Merkle Tree, mantido salvo no computador a Merkle Root, para posterior verificação. Ao baixar um desses arquivos, solicitamos que venha junto o conjunto de hashes necessários para computar uma nova raiz, conhecido como Merkle Proof.

Em posse do nosso arquivo, calculamos o seu hash, e em conjunto com a Merkle Proof, geramos uma nova Merkle Tree. A verificação é feita comparando a raiz previamente salva com a raiz da nova árvore gerada, se qualquer modificação tiver sido feita no arquivo, ou este tenha sido corrompido, seu hash não será o mesmo, consequentemente a raiz não terá como ser igual. Então, nesse cenário, se as duas raízes forem iguais, provamos a integridade do arquivo, e sua presença na geração da primeira Merkle Tree.

3) Utilização

Merkle trees são muito úteis por usarem hashes ao invés de arquivos inteiros para garantir integridade, garantindo assim uma grande economia de espaço. Seus principais usos visam checar a integridade de arquivos, garantir a existência de uma transação entre dois pares, e transmitir arquivos em canais não-confiáveis. É um algoritmo altamente usado em sistemas distribuídos para uma verificação de dados mais eficiente, e conexões P2P como Tor, e no GIT.

Este sistema é também muito utilizado na implementação de criptomoedas, tal como o bitcoin, para garantir a presença de uma transação dentro de um bloco e garantir a integridade dos mesmos.

2. Código

Como mencionado no tópico anterior, nossa implementação tem como objetivo completar o código para geração da Merkle Proof, conjunto de hashes necessários para calcular uma nova Merkle Root a partir de uma determinada folha.

Foram previamente fornecidos os códigos *verifier.py*, completo e *prover.py* - para ser completado. Vamos focar em explicar a implementação feita na função *gen_merkle_tree* mas também serão explicadas superficialmente as outras funções presentes no código.

2.1 Prover.py

Código responsável pela geração da merkle tree, seu funcionamento acontece tal que são geradas 1000 strings aleatórias, as folhas da árvore, é calculada a prova merkle para uma dessas folhas e então gerado um arquivo contendo a prova, contendo as informações sobre a folha estudada, sua posição e a prova calculada.

1) **Funções**

- ***gen_merkle_proof***

A função *gen_merkle_proof*, como dito no comentário que a descreve, recebe como input uma lista de folhas de uma árvore e a posição da folha a ser testada e retorna a merkle proof para a mesma.

variáveis relevantes

- **level**: nível atual referente à altura da árvore sendo percorrida
- **level_pos**: posição do elemento sendo verificado dentro da lista de hashes (state, representando lista dos elementos do andar atual da árvore)
- **state**: lista de hashes, inicialmente das folhas, posteriormente da árvore merkle
- **path_to_add**: variável que guarda elemento a ser adicionado no caminho da merkle proof
- **path:** lista completa dos hashes presentes na merkle proof

funcionamento

1 - Constrói a base da árvore, as folhas, criando seus hashes, armazenando em *state* e manipulando-a para ter um número de folhas n tal que exista um x em que log2x = n.

2 - Entra num loop para percorrer todos os andares da árvore, e para cada andar:

a) Calcula localização da posição recebida dentro da árvore, para saber se o elemento companheiro de operação será o da esquerda ou da direita
b) Adiciona o elemento encontrado no caminho final da *merkle proof*
c) Calcula o andar de cima da árvore, utilizando *list slice* para separar os elementos da esquerda e da direita, e a função

fornecida *hash_internal_node,* explicada no item (item que explica)

d) Atualiza o *state* para o andar superior da árvore, e o *level_pos* para a posição que representa o nó pai do elemento atual

3 - Retorna lista de hashes armazenada em path

- ***hash_leaf***

  Responsável por gerar hash de uma folha recebida

  variáveis relevantes

  - **leaf**: conjunto de bytes representando uma folha

  funcionamento

  Cria e retorna hash com informação em *leaf*

- ***hash_internal_node***

  Recebe dois elementos, o da esquerda e o da direita, e retorna o hash do elemento resultante, correspondente ao nó pai dos dois elementos fornecidos

  variáveis relevantes

  - **left**: hash correspondente à folha da esquerda
  - **right**: hash correspondente à folha da direita

  funcionamento

  Cria e retorna um novo hash a partir da combinação de *left* e *right*

- ***write_proof***

  Gera arquivo final, nomeado ‘proof.txt’ no mesmo diretório do código, formatando os valores tal que o arquivo tenha este formato

2) Código da parte implementada

```
def gen_merkle_proof(leaves, pos):
    """Takes as input a list of leaves and a leaf position (pos).
       Returns the Merkle proof for the leaf at pos."""

    height = math.ceil(math.log(len(leaves),2))
    assert height < MAXHEIGHT, "Too many leaves."

    # hash all the leaves
    state = list(map(hash_leaf, leaves))

    # Pad the list of hashed leaves to a power of two
    padlen = (2**height)-len(leaves)
    state += [b"\x00"] * padlen

    # initialize a list that will contain the hashes in the proof
    path = []

    level_pos = pos    # local copy of pos


    "Parte implementada por Amanda Costa"
    for level in range(height):

        # encontra localização da posição na tree (direita/esquerda)
        if level_pos % 2 == 0:
            path_to_add = state[level_pos + 1]
        else:
            path_to_add = state[level_pos - 1]

        # adiciona hash da folha relacionada ao path de prova
        path.append(path_to_add)

        # computa o nível novo da merkle tree (andar acima)
        left = state[::2]
        right =  state[1::2]
        state = list(map(hash_internal_node, left, right))

        # sobe um nível na árvore - posição agora corresponde ao nó pai
```

```
    level_pos = level_pos//2

# return a list of hashes that makes up the Merkle proof
return path
```

2.2 Verifier.py

Código responsável pela verificação da merkle proof gerada no arquivo anterior, dentro do seu escopo, é fornecida uma merkle root, correspondente à da árvore gerada em prover.py. No seu funcionamento, acontece a leitura do arquivo proof.txt (gerado por *write_proof*), é gerada então uma merkle root com a prova fornecida, e, finalmente, esta é comparada à raiz pré-existente, verificando assim a veracidade da prova.

1) **Funções**

- ***read_proof***
  Realiza leitura do texto presente no arquivo proof.txt e armazena os valores encontrados.

  funcionamento
  Após realizar a leitura do arquivo e atribuir seus valores a variáveis, salva as informações num objeto MerkleProof, que possui os dados da folha sendo checada, a posição da mesma, e o caminho de hashes fornecido para os cálculos, das folhas até a raiz.

- ***hash_leaf***
  Semelhante à *hash_leaf* de *prover.py*

- ***hash_internal_node***
  Semelhante à *hash_internal_node* de *prover.py*

- ***compute_merkle_root_from_proof***
  Função responsável por calcular a raiz com a prova fornecida, funciona de uma maneira parecida com *gen_merkle_proof*

  variáveis relevantes

  - **pos**: posição do elemento sendo verificado dentro da lista de hashes

- **path**: lista completa dos hashes presentes na merkle proof
- **root**: variável utilizada para salvar o hash do elemento sendo testado

funcionamento
1 - salva a posição, os hashes do caminho da prova e considera a folha fornecida posição inicial
2 - Entra num loop para percorrer todos os hashes da prova e em cada um:
   a) Calcula localização da posição recebida dentro da árvore (direita/esquerda)
   b) Calcula o nó pai do elemento fornecido com o elemento irmão encontrado no item a
   c) Considera o nó pai a nova raiz e sobe um nível na árvore para prosseguir com os cálculos
3 - Finalmente, retorna a raiz computada

3. Considerações Finais

Neste relatório abordamos a implementação de uma Merkle Proof, assim como entendimento sobre o conceito de Merkle Trees e sua relevância na garantia de integridade nos projetos em que é usada.

Sobre a implementação de fato, embora tenhamos tido pouca interferência no funcionamento do código em geral, sabendo que só fizemos uma função de algo pré-implementado, foi uma experiência muito interessante, e criou o interesse de aprofundamento no assunto para uma implementação própria completa no futuro.

4. Bibliografia

Merkle Tree. In Wikipedia. 2018. Disponível em https://en.wikipedia.org/wiki/Merkle_tree Acesso em outubro de 2022.

Merkle Tree, In Brilliantorg. Disponível em https://brilliant.org/wiki/merkle-tree/. Acesso em outubro de 2022.

What Is a Merkle Tree. In Decentralized Thoughts, 2020. Disponível em https://decentralizedthoughts.github.io/2020-12-22-what-is-a-merkle-tree/. Acesso em setembro de 2022.