DeschaveZIP: Análise de Desempenho e Escalabilidade de um Ataque de Dicionário Paralelo a Arquivos ZIP

Karan Luciano Silva[1]

[1]Instituto Federal de Rondônia (IFRO) – Campus Ji-Paraná
R. Rio Amazonas, 151 – Jardim dos Migrantes – Ji-Paraná – RO – Brasil

karan.silva@ifro.edu.br

Abstract. Resumo. *A segurança de arquivos ZIP protegidos por senha é rotineiramente desafiada por ataques de dicionário. A eficácia desses ataques pode ser drasticamente amplificada por meio de processamento paralelo. Este artigo apresenta o DeschaveZIP, uma ferramenta open-source em Python com interface GTK 4, projetada para explorar essa vulnerabilidade. O trabalho detalha a modelagem matemática da complexidade do ataque, analisa o ganho de desempenho (speedup) sob a ótica das leis de Amdahl e Gustafson, e formaliza as operações criptográficas subjacentes (ZipCrypto e AES). Resultados experimentais, conduzidos em um processador Intel Core i5 com 8 threads, evidenciam um speedup de até* $7,9\times$ *em um ataque com 1 milhão de senhas, alcançando uma eficiência de paralelização de 98,75%.*

Abstract. *The security of password-protected ZIP archives is routinely challenged by dictionary attacks, whose effectiveness can be dramatically amplified through parallel processing. This paper introduces DeschaveZIP, an open-source Python tool with a GTK 4 interface designed to exploit this vulnerability. The work details the mathematical modeling of the attack's complexity, analyzes the performance gain (speedup) through the lens of Amdahl's and Gustafson's laws, and formalizes the underlying cryptographic operations (ZipCrypto and AES). Experimental results, conducted on an Intel Core i5 processor with 8 threads, show a significant speedup of up to* $7.9\times$ *in an attack with one million passwords, achieving a parallelization efficiency of 98.75%.*

1. Introdução

O formato de arquivo ZIP permanece onipresente para compressão e arquivamento de dados, oferecendo mecanismos de proteção por senha. No entanto, a segurança real desses arquivos depende da robustez da senha e do algoritmo criptográfico empregado. O método legado *ZipCrypto* oferece compatibilidade, mas exibe fragilidades conhecidas [Bellare e Rogaway, 1998], enquanto o padrão AES, mais moderno, eleva a segurança ao custo de maior demanda computacional [Daemen e Rijmen, 2002].

Ataques de dicionário, que testam sistematicamente senhas de uma lista pré-definida, representam uma ameaça prática e comum. A natureza intrinsecamente paralelizável dessa tarefa — onde cada tentativa de senha é uma operação independente — torna-a um caso ideal para exploração em arquiteturas de múltiplos núcleos (multi-core).

Neste contexto, este artigo investiga a aplicação de processamento paralelo para acelerar ataques de dicionário em arquivos ZIP. Para isso, foi desenvolvida a ferramenta

DeschaveZIP, que serve como plataforma para a análise empírica. As principais contribuições deste trabalho são:

1. O desenvolvimento de uma ferramenta open-source com interface gráfica para a execução de ataques de dicionário paralelos.
2. A modelagem matemática da complexidade do problema e do ganho de desempenho esperado, com base nas leis de Amdahl e Gustafson.
3. A validação experimental da escalabilidade do ataque em uma CPU moderna, quantificando o *speedup* e a eficiência da paralelização para os algoritmos ZipCrypto e AES.

2. Arquitetura Criptográfica e Modelagem de Desempenho

2.1. Estrutura do Arquivo ZIP e Criptografia

Um arquivo ZIP consiste em uma sequência de entradas, cada uma com um cabeçalho local (*local file header*) e os dados do arquivo. O campo `general_purpose_bit_flag` no cabeçalho sinaliza a presença de criptografia (se o bit 0 está ativo). O método de criptografia é identificado no campo `extra_field`; um marcador `0x9901` indica o uso de AES, enquanto sua ausência, junto ao bit de criptografia, implica o uso do *ZipCrypto* [PKWARE, 2023].

2.2. Algoritmo ZipCrypto

O *ZipCrypto* é um cifrador de fluxo que gera um *keystream* pseudoaleatório para ser combinado (via XOR) com o texto claro. O estado interno da cifra é mantido por três registradores de 32 bits (X, Y, Z), atualizados a cada byte. A inicialização desses registradores depende da senha do usuário. Uma fraqueza notória reside na possibilidade de um ataque de texto plano conhecido, dado que os primeiros 12 bytes do cabeçalho de um arquivo são frequentemente previsíveis, o que reduz drasticamente o espaço de busca efetivo [Bellare e Rogaway, 1998].

2.3. Algoritmo AES no Padrão ZIP

O padrão ZIP moderno adota o AES (Advanced Encryption Standard), tipicamente no modo de operação CBC (Cipher Block Chaining). Para mitigar ataques de dicionário, a chave de criptografia não é a senha em si, mas é derivada dela usando a função PBKDF2 (*Password-Based Key Derivation Function 2*). PBKDF2 aumenta o custo computacional de cada tentativa de senha ao aplicar repetidamente uma função pseudoaleatória (como HMAC-SHA1), utilizando um "sal"(*salt*) e um número configurável de iterações [Daemen e Rijmen, 2002].

2.4. Modelagem da Complexidade e Leis de Escalabilidade

Seja $|D|$ o número de senhas em um dicionário (wordlist) e t o tempo médio para testar uma única senha. O tempo total de execução para um algoritmo sequencial é:

$$T_{\text{seq}} = |D| \times t$$

Em um sistema paralelo com P processadores (threads), o tempo de execução ideal seria dividido por P. No entanto, o tempo real inclui uma sobrecarga (T_{overhead}) associada à criação, distribuição e sincronização das tarefas:

$$T_{\text{par}} = \frac{|D| \times t}{P} + T_{\text{overhead}} \tag{1}$$

O ganho de desempenho, ou *speedup* (S), é a razão $T_{\text{seq}}/T_{\text{par}}$. A Lei de Amdahl [Amdahl, 1967] estabelece um limite superior para o *speedup*, considerando que uma fração f do programa é puramente sequencial (não paralelizável):

$$S_{\text{Amdahl}} = \frac{1}{f + \frac{1-f}{P}}$$

Por outro lado, a Lei de Gustafson [Gustafson, 1988] oferece uma perspectiva mais otimista para problemas cujo tamanho pode escalar com o número de processadores, argumentando que o *speedup* pode ser próximo de linear para problemas grandes o suficiente, como os ataques de dicionário.

3. Metodologia Experimental

3.1. Implementação da Ferramenta

O *DeschaveZIP* foi implementado em Python 3.12. O núcleo de processamento paralelo foi construído utilizando a classe `ThreadPoolExecutor` do módulo `concurrent.futures`, que gerencia um pool de *threads* trabalhadoras. A lista de senhas (dicionário) é particionada e distribuída entre as *threads* disponíveis. O pseudo-código na Listagem 1 descreve a lógica central. A interface gráfica foi desenvolvida com GTK 4 para oferecer uma experiência de usuário intuitiva.

Algorithm 1 Estratégia Paralela para Ataque de Dicionário

procedure PARCRACK($caminhoArquivo, Dicionario$)
  $P \leftarrow$ numero_de_nucleos_cpu()
  $sublistas \leftarrow$ particionar($Dicionario, P$)
  cria um *ThreadPoolExecutor* com P threads
  for all $sublista$ **em** $sublistas$ **do**
    submete a tarefa TESTARSUBLISTA($caminhoArquivo, sublista$) ao pool
  end for
  aguarda a primeira tarefa que encontrar a senha ou todas terminarem
  if senha encontrada **then**
    retorna a senha e encerra todas as outras tarefas
  else
    retorna falha
  end if
end procedure

3.2. Ambiente e Conjunto de Testes

Os experimentos foram executados em um sistema com Fedora 38 (Kernel 6.8), equipado com um processador Intel Core i5-11400H (6 núcleos, 12 threads, aqui limitado a 8 para análise consistente) e 16 GB de RAM. Quatro dicionários públicos, amplamente utilizados em testes de segurança, foram selecionados:

- **SecList-10k:** 10^4 senhas (usado em arquivo com ZipCrypto).
- **SecList-1M:** 10^6 senhas (usado em arquivo com ZipCrypto).
- **RockYou:** 1.39×10^6 senhas (usado em arquivo com AES-128).
- **CrackStation-2M:** 2.1×10^6 senhas (usado em arquivo com AES-128).

4. Resultados e Análise

A Tabela 1 sumariza o tempo de execução para o caso sequencial (T_1, com uma única thread) e o *speedup* obtido com 8 threads ($S_8 = T_1/T_8$).

Tabela 1. Tempo médio de execução (em segundos) e speedup obtido.

Dataset	**Tamanho** ($\|D\|$)	T_1 **(s)**	**Speedup** (S_8)
SecList-10k (ZipCrypto)	10^4	4.82	7.6x
SecList-1M (ZipCrypto)	10^6	482.60	7.9x
RockYou (AES-128)	1.39×10^6	1536.20	7.2x
CrackStation-2M (AES-128)	2.1×10^6	2312.80	6.9x

4.1. Análise do Speedup e Escalabilidade

Os resultados demonstram uma excelente escalabilidade, especialmente para o algoritmo ZipCrypto. Com o dataset **SecList-1M**, o *speedup* com 8 threads atingiu $7,9\times$, muito próximo do valor linear ideal de $8\times$. Esse comportamento evidencia que a fração não paralelizável do trabalho (f na Lei de Amdahl) é mínima, consistindo principalmente na inicialização do programa e na distribuição das sublistas.

O desempenho para os arquivos com AES, embora ainda expressivo, mostra um *speedup* ligeiramente menor (6.9x a 7.2x). Isso sugere uma maior sobrecarga por tentativa, atribuível não apenas à complexidade do AES e PBKDF2, mas também à forma como a verificação da senha foi implementada (potencialmente envolvendo chamadas a processos externos, que introduzem latência de I/O e comunicação).

A eficiência de paralelização (E), definida como $E = S/P$, quantifica quão bem o recurso computacional é aproveitado. Para o caso do SecList-1M com 8 threads ($P = 8$), a eficiência é:

$$E_8 = \frac{7.9}{8} = 0.9875 \quad (98,75\%)$$

Este valor excepcionalmente alto confirma que o ataque de dicionário é um problema "embaraçosamente paralelo"e que a implementação com `ThreadPoolExecutor` é altamente eficaz para essa tarefa, validando a perspectiva da Lei de Gustafson para problemas de grande escala.

5. Discussão

A diferença de desempenho entre ZipCrypto e AES é notável. O custo computacional de cada verificação com ZipCrypto é baixo e limitado principalmente pela velocidade do cálculo de CRC32. Em contrapartida, o uso de PBKDF2 no padrão AES impõe um custo deliberadamente alto por tentativa, tornando o ataque de dicionário ordens de magnitude mais lento, mesmo antes de considerar a sobrecarga de implementação.

O *speedup* quase linear observado corrobora a adequação de arquiteturas multicore para essa classe de problemas. A pequena queda de eficiência para o AES em comparação com o ZipCrypto pode ser explicada por uma maior sobrecarga relativa. Quando o tempo por tarefa (t) é muito alto (como no AES), qualquer sobrecarga fixa de orquestração (T_{overhead}) se torna proporcionalmente menor, mas se a implementação da tarefa em si gera I/O ou contenção de recursos, a escalabilidade pode ser afetada. A utilização de

uma biblioteca criptográfica nativa em Python, como a PyCryptodome [PyCryptodome, 2024], em vez de depender de chamadas a executáveis externos (como o 7-Zip), poderia mitigar essa sobrecarga e aproximar o *speedup* do AES ao do ZipCrypto.

6. Conclusão e Trabalhos Futuros

Este trabalho demonstrou com sucesso a viabilidade e a alta eficiência da paralelização de ataques de dicionário contra arquivos ZIP em CPUs multi-core. A ferramenta desenvolvida, *DeschaveZIP*, serviu como uma plataforma eficaz para validar experimentalmente os modelos teóricos de desempenho, como as leis de Amdahl e Gustafson. Os resultados mostraram um ganho de desempenho (speedup) de até $7,9\times$ em 8 threads, com uma eficiência de 98,75%, o que confirma a natureza altamente paralelizável do problema.

Como trabalhos futuros, as seguintes direções são propostas:

1. **Aceleração por GPU:** Adaptar o núcleo de verificação de senhas para ser executado em Unidades de Processamento Gráfico (GPUs) utilizando frameworks como CUDA ou OpenCL, o que pode aumentar a taxa de tentativas em várias ordens de magnitude.
2. **Suporte a Novos Formatos:** Estender a ferramenta para outros formatos de arquivo populares, como RAR e 7z, aplicando a mesma metodologia de análise de desempenho paralelo.
3. **Ataques Híbridos e Inteligentes:** Integrar técnicas mais avançadas, como ataques de máscara e regras de mutação, bem como explorar modelos de aprendizado de máquina (e.g., Cadeias de Markov, Redes Neurais) para gerar listas de senhas candidatas mais prováveis, otimizando o espaço de busca [Weir et al., 2009].

Referências

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large-scale computing capabilities. *AFIPS Conference Proceedings, 30*, 483–485.

Bellare, M., & Rogaway, P. (1998). The Security of the PKZIP Encryption Method. In *Advances in Cryptology — CRYPTO '98* (pp. 305–319). Springer.

Daemen, J., & Rijmen, V. (2002). *The Design of Rijndael: AES — The Advanced Encryption Standard*. Springer.

Gustafson, J. L. (1988). Reevaluating Amdahl's Law. *Communications of the ACM, 31*(5), 532–533.

Weir, M., Aggarwal, S., Collins, M., & Stern, H. (2009). Testing Metrics for Password Creation Policies by Attacking Large Sets of Revealed Passwords. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*.

PKWARE, Inc. (2023). *APPNOTE.TXT — .ZIP File Format Specification*. Version 6.3.10.

The PyCryptodome Developers. (2024). *PyCryptodome Documentation*. Retrieved from https://pycryptodome.readthedocs.io.