APKHash: Grafos de Similaridade Estrutural como Evidência de Ruído na Rotulação de Famílias de Malware

Felipe Duarte[1], Eloiza Rossetto[1], João Pincovscy[2,3], Paulo Lisboa de Almeida[1], André Grégio[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
SecRET (secret.seg.br) – Curitiba – PR – Brasil
{felipeduarte, eloiza.rossetto, paulorla, gregio}@ufpr.br

[2]Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações
[3]UnDF - Universidade do Distrito Federal
Brasília – DF – Brasil
pincovscy@cepesc.gov.br; joao.pincovscy@undf.edu.br

Abstract. *In this work, we present APKHash, an approach based on opcode n-grams, MinHash/LSH, and similarity graphs for the structural analysis of Android applications. The methodology was evaluated on 12,000 APKs from 24 malware families collected between 2022 and 2024. The results show that heterogeneous components persist even in restrictive settings, indicating shared structural patterns between families or inconsistencies in dataset labeling. Regarding performance, the use of MinHash/LSH as APKHash main approach reduced the search space by approximately 97%. APKHash captures relevant structural relationships and reveals strong intra-family cohesion, but also limitations in separating malware families. The approach proves useful for structural analysis and for diagnosing the quality of large-scale malware datasets labeling.*

Resumo. *Neste trabalho, apresentamos o APKHash, uma abordagem baseada em n-gramas de opcodes, MinHash/LSH e grafos de similaridade para a análise estrutural de aplicações Android. A metodologia foi avaliada em 12.000 APKs de 24 famílias de malware coletadas entre 2022 e 2024. Os resultados mostram que componentes heterogêneos persistem mesmo em configurações restritivas, indicando padrões estruturais compartilhados entre famílias ou inconsistências na rotulagem do conjunto de dados. Em relação ao desempenho, o uso de MinHash/LSH como principal abordagem do APKHash reduziu o espaço de busca em aproximadamente 97%. O APKHash captura relações estruturais relevantes e revela forte coesão intra-família, mas também limitações na separação entre famílias de malware. A abordagem mostrou-se útil tanto para a análise estrutural quanto para o diagnóstico da qualidade da rotulagem de grandes conjuntos de dados de malware.*

1. Introdução

A identificação de famílias de *malware* é uma etapa importante para a área de cibersegurança. Embora o foco geral se volte para a detecção de *malware*, somente a partir do estudo de exemplares é possível compreender os mecanismos explorados por atacantes e criar defesas mais eficazes para sistemas computacionais. Uma família de *malware* pode

ser definida como uma coleção de arquivos maliciosos originários de uma mesma base de código [Joyce et al. 2025]. Determinar a família de um exemplar ou definir novas famílias é um trabalho feito manualmente por analistas de segurança. Após a análise, um "nome"e uma assinatura são criados e posteriormente usados na indústria por antivírus.

Esse processo é oneroso e há conhecimento da área que a rotulação de famílias é muitas vezes inconsistente, como destacado por [Wang et al. 2024]. Não há um consenso de rótulo entre diferentes antivírus e variantes surgem tão rapidamente que analistas não conseguem atualizar assinaturas no mesmo passo. Atualmente, com a popularização e o uso extensivo de algoritmos de aprendizado de máquina, a extração de características para a criação de conjuntos de dados vem se tornando cada vez mais importante. A qualidade dos conjuntos de dados usados para treinamento e validação impacta diretamente a qualidade final dos modelos gerados e, portanto, na capacidade de proteger sistemas computacionais. Nesse cenário, este artigo propõe-se agrupar famílias de *malware* de forma mais representativa quanto a sua similaridade fornecendo insights sobre a qualidade dos rótulos dos exemplares atribuídos a uma dada família.

Para tanto, introduz-se o APKHash, um *pipeline* para representação de APKs baseado em um grafo de similaridade construído a partir de dados dos arquivos `.dex`. Cada APK é representada por padrões locais extraídos com n-gramas refinados e transformados em assinaturas utilizando MinHash. Em seguida, emprega-se *Locality Sensitive Hashing* (LSH) [Paulevé et al. 2010] para agrupar assinaturas semelhantes sem a necessidade de comparar todas as APKs diretamente. O resultado é um grafo de similaridade no qual cada nó representa uma APK e as arestas conectam aplicações cuja similaridade de Jaccard ultrapassa um limiar definido. Como esse grafo é naturalmente desconexo, surgem componentes conectados que podem ser analisados para investigar relações estruturais entre exemplares e validar os rótulos de famílias de *malware*.

O APKHash foi avaliado em um subconjunto do AndroZoo [Allix et al. 2016], utilizando as famílias definidas em [Hurier et al. 2017]. A partir desses dados, foram aplicados filtros para construir um conjunto balanceado contendo apenas famílias com pelo menos 500 amostras, resultando em um *dataset* com 12.000 APKs distribuídos em 24 famílias. Sobre o grafo gerado, conduzimos uma análise dos principais componentes conectados e investigamos relações estruturais entre famílias de *malware* a partir das representações produzidas pelo APKHash.

Os resultados indicam que o método é capaz de gerar representações discriminativas para diferentes famílias de *malware*. A análise dos componentes conectados revelou a presença de famílias com assinaturas mais genéricas, além de possíveis inconsistências nos rótulos atribuídos, sugerindo erros de rotulagem ou sobreposição comportamental entre amostras de *malware*.

Portanto, as principais contribuições trazidas por esse trabalho são: (i) APKHash, um método escalável para a geração de representações estruturais de APKs voltado à análise de similaridade entre aplicações Android; (ii) Análise de componentes gerados pelo método para a validação de rótulos de família de *malwares* Android; (iii) Discussão sobre o comportamento e relação entre famílias de *malware* Android.

2. Trabalhos Relacionados

Na literatura, diferentes trabalhos investigam similaridade entre aplicações Android utilizando informações extraídas do APK, como permissões, chamadas de API, estrutura de classes ou características do bytecode. Entre essas abordagens, os trabalhos [Canfora et al. 2015], [Kang et al. 2016] e [Lee et al. 2019] utilizam n-gramas de *opcodes* extraídos dos arquivos `.dex` para representar aplicações Android. Em [Canfora et al. 2015], os autores analisam a eficácia de sequências de *opcodes* para detecção de *malware*s de múltiplas famílias, mostrando que padrões locais do bytecode possuem capacidade discriminativa. De forma semelhante, [Kang et al. 2016] combina n-gramas de *opcodes* com aprendizado de máquina para classificação de *malware* Android, enquanto o *Dexofuzzy* [Lee et al. 2019] utiliza *fuzzy hashing* sobre sequências de *opcodes* para detectar variantes de *malware*.

Outros trabalhos investigam comparação de aplicações Android completas para detectar clonagem, reutilização de código, variantes ou *repackaging*. O trabalho *AnDarwin* propõe uma abordagem escalável baseada em informação semântica e MinHash/LSH para evitar comparações par a par entre APKs [Crussell et al. 2015]. Já o *RepDetector* [Guan et al. 2016] utiliza relações de entrada e saída entre funções para detectar *repackaging*, buscando robustez frente a transformações que preservam semântica. Trabalhos mais recentes, como [De Ghein et al. 2022] e [Dai et al. 2025], investigam comparação estrutural entre versões de aplicações Android por meio da estrutura de classes e técnicas de *diffing*, permitindo identificar correspondências mesmo na presença de obfuscação. Além disso, técnicas de *similarity hashing* também têm sido utilizadas para detecção de variantes de *malware* Android. O *FLSH* utiliza *fuzzy hashing* sobre características estáticas de APKs para detecção de *malware* e variantes [Hadi et al. 2025].

Há também trabalhos que utilizam grafos e redes de similaridade para modelar relações entre aplicações Android. Em [Frenklach et al. 2021a] e [Frenklach et al. 2021b], grafos de similaridade entre APKs são empregados para apoiar a detecção de *malware*. Já em [Karbab et al. 2020], o *Cypider* constrói uma infraestrutura baseada em redes de similaridade e aplica detecção de comunidades para particionar amostras de *malware* Android. De forma semelhante, [Kim et al. 2019] utiliza redes de similaridade e análise de comunidades para classificar famílias de *malware*. *Frameworks* como o *MATRIX* [Simoni et al. 2025] também utilizam representações baseadas em grafos para apoiar análise de *malware* e investigação de ameaças em larga escala. Trabalhos recentes, como *DeepCatra* [Wu et al. 2023], [Li et al. 2024] e *GHGDroid* [Shen et al. 2024], utilizam grafos heterogêneos, fluxo de execução e características extraídas do bytecode para tarefas de detecção de *malware* Android.

Em geral, a literatura apresenta técnicas para detectar *repackaging*, comparar versões ou aplicações, aplicar *similarity hashing* e utilizar grafos em tarefas relacionadas a *malware* Android. Diferente dessas abordagens, o APKHash combina n-gramas de *opcodes*, MinHash/LSH e grafos de similaridade para analisar como APKs Android se organizam estruturalmente em escala. A representação proposta permite estudar agrupamento, fragmentação e mistura entre famílias de *malware*, possibilitando investigar relações estruturais entre APKs independentemente dos rótulos atribuídos aos conjuntos de dados.

3. Metodologia e Conceitos

Nessa seção, será discutida a metodologia utilizada no APKHash, apresentando-se os conceitos necessários em cada etapa de aplicação da mesma.

3.1. O Conjunto de Dados

Aplicações Android são distribuídas no formato APK, que possui diferentes componentes necessários para a execução da aplicação, como códigos, manifesto, metadados, etc. Dentre esses componentes, os arquivos `.dex` armazenam o *bytecode* Dalvik da aplicação, que servem como entrada para análise do APKHash, pois eles possuem a representação do código executável de apps Android. Essa escolha de representação torna a comparação mais focada no código executável, mas também impõe limites relacionados ao código nativo, recursos, manifesto e comportamento dinâmico não serem integrados na análise. Por outro lado, bibliotecas comuns, empacotadores (ferramentas que comprimem, otimizam e protegem o código), e trechos reutilizados podem estar presentes em diferentes APKs e produzir similaridade entre aplicações que possuem rótulos de famílias diferentes, possibilitando o agrupamento pela estrutura da APK e auxiliando a investigação de rotulação errônea.

APKHash foi avaliado com parte do *dataset* AndroZoo [Allix et al. 2016], um dos maiores conjuntos de dados de *malware* de android disponíveis na literatura. O AndroZoo conta com milhões de APKs benignas e maliciosas. Em [Hurier et al. 2017], os autores selecionam um subconjunto do Androzoo e criam um método (Euphony) para unificar rótulos de famílias de *malware* a partir de detecções de diferentes antivírus. Assim, os autores disponibilizam um *dataset* com mais de 200 mil amostras rotuladas via Euphony.

A partir dessas informações, foi feito o *download* das amostras, as quais foram passadas para uma plataforma de análise de APKs [Júnior et al. 2025]. Essa etapa foi crucial para garantir que as APKs fossem completas e funcionais. Durante o processo de execução dinâmica, muitos exemplares foram descartados por se tratarem apenas de fragmentos de arquivos executáveis. Em diversos casos, não foi possível reconstruir as APKs a partir desses fragmentos. Assim, optou-se pelo descarte dessas amostras, uma vez que impossibilitaria a comparação com APKs completas.

Ao todo, obteve-se 42.230 APKs analisadas dinamicamente, distribuídas em 54 famílias diferentes. Em seguida, aplicamos um novo filtro aos dados, mantendo apenas as famílias com pelo menos 500 exemplares, o que resultou em 24 famílias selecionadas. Além disso, foi estabelecido um limite de 500 APKs por família, com o objetivo de obter uma base de dados balanceada para os experimentos. Dessa forma, a base final é composta por 12.000 amostras pertencentes a 24 famílias, as quais serão listadas após aceite para reprodução dos experimentos feitos neste artigo.

3.2. Extração de n-gramas

A partir dos arquivos `.dex`, são extraídas sequências de *opcodes*, os quais representam a operação executada por uma instrução de *bytecode* como: operações de carga e movimentação, (mover valores literais ou dados entre registradores), operações aritméticas e lógicas, operações de objetos e campos, entre outras. Os *opcodes* são então transformados em 4-gramas, que representam sequências locais compostas por quatro operações consecutivas do *bytecode* da APK. Essa representação mantém a ordem local das instruções e

permite capturar padrões estruturais do código, evitando comparar a sequência completa de *opcodes* que pode ser mais sensível a pequenas modificações no *bytecode*.

Cada n-grama pode ser representado por um identificador numérico, reduzindo o custo de armazenamento e comparação. Dessa forma, cada APK passa a ser tratada como um conjunto de identificadores derivados dos seus n-gramas, o que permite aplicar medidas de similaridade entre conjuntos, como Jaccard, e técnicas de indexação aproximada, como MinHash e LSH.

Por fim, os 4-gramas extraídos são deduplicados para capturar a presença de padrões locais no arquivo, fazendo com que cada APK passe a ser representada por um conjunto quadrigramas únicos. Após a extração, cada conjunto gerado é serializado para reduzir o tamanho de armazenamento utilizando o BLAKE2b [Saarinen and Aumasson 2015]. Com essa etapa cada elemento do conjunto passa a ocupar o tamanho fixo de 8 *bytes*.

3.3. Assinaturas MinHash e Indexação LSH

A comparação exata entre todos os pares de elementos de uma base cresce quadraticamente com o número de amostras, o que pode se tornar inviável para bases grandes. Técnicas como MinHash e *Locality-Sensitive Hashing* (LSH) são utilizadas para reduzir esse custo, recuperando pares candidatos que possuem maior probabilidade de serem similares.

O MinHash é uma técnica utilizada para representar conjuntos por meio de assinaturas compactas. A ideia central é aproximar a similaridade de Jaccard entre conjuntos: quando dois conjuntos compartilham muitos elementos, suas assinaturas MinHash tendem a ser mais parecidas. Dessa forma, a comparação pode ser feita sobre assinaturas menores, em vez de exigir inicialmente a comparação completa entre todos os elementos dos conjuntos originais.

O LSH é utilizado para organizar essas assinaturas de modo que itens semelhantes tenham maior chance de cair nos mesmos grupos de busca, chamados *buckets*. Quando duas amostras aparecem no mesmo *bucket*, elas são tratadas como pares candidatos para uma comparação posterior mais precisa. Assim, o LSH não define sozinho a similaridade final entre duas amostras, mas reduz o espaço de busca para os pares mais promissores.

Portanto, para reduzir o custo das comparações exaustivas na geração de pares de APKs similares, o APKHash utiliza assinaturas MinHash. Cada conjunto de n-gramas extraído das APKs é transformado em uma assinatura de tamanho fixo. Logo, em vez de comparar diretamente os conjuntos completos de n-gramas, o APKHash realiza comparações entre assinaturas compactas, reduzindo o custo computacional. Nos experimentos, foi utilizado com configuração 256 permutações MinHash e semente aleatória 42.

A partir das assinaturas MinHash, aplicou-se a técnica de *Locality-Sensitive Hashing* (LSH) para recuperar pares candidatos de forma eficiente. Para isso, cada assinatura é dividida em b bandas contendo r linhas cada, de modo que $b \cdot r = k$, em que k representa o número total de permutações MinHash. Dois APKs são considerados candidatos quando apresentam coincidência em pelo menos uma das bandas.

Nos experimentos, utilizou-se b=32 e r=8, reduzindo os 71.994.000 pares possíveis a 1.886.845 candidatos únicos, uma redução de 97,38%.

3.4. Construção do Grafo de Similaridade

A partir dos pares candidatos recuperados por LSH, é construído um grafo de similaridade $G = (V, E)$, em que cada nó $v \in V$ representa um APK. Uma aresta $(a, b) \in E$ é adicionada quando o par (a, b) é recuperado pelo LSH e sua similaridade de Jaccard é maior ou igual a um limiar t:

$$(a, b) \in E \iff (a, b) \text{ é candidato LSH} \wedge J(G_a, G_b) \geq t.$$

Foram avaliados os limiares $t \in \{0{,}5, 0{,}6, 0{,}7, 0{,}8, 0{,}9, 1{,}0\}$, abrangendo configurações progressivamente mais restritivas. O valor $t = 0{,}8$ foi selecionado para o estudo de caso por apresentar o maior F1 pareado, enquanto $t = 1{,}0$ foi utilizado como cenário extremo, no qual apenas representações idênticas são conectadas.

O grafo de similaridade gerado pelo APKHash é naturalmente desconexo, isto é, não existe necessariamente um caminho entre todos os pares de vértices do grafo. Dessa forma, parte das análises realizadas considera seus componentes conectados, definidos como subgrafos maximais $C = (V_C, E_C) \subseteq G$ nos quais $\forall u, v \in V_C,\ \exists\, p(u, v)$.

Além disso, os componentes conectados são classificados de acordo com os rótulos de família dos APKs. Seja $f(v)$ a função que retorna a família associada ao vértice v. Um componente é considerado homogêneo quando $|\{f(v) \mid v \in V_C\}| = 1$, enquanto componentes heterogêneos satisfazem $|\{f(v) \mid v \in V_C\}| > 1$.

Essa distinção permite analisar o grau de coesão estrutural entre APKs de uma mesma família, bem como identificar relações estruturais compartilhadas entre famílias distintas.

4. Experimentos e Resultados

Nesta seção são apresentados e discutidos os resultados obtidos com a aplicação do APKHash sobre o conjunto de malware Android analisado. Inicialmente, são avaliados os pares candidatos recuperados pelo MinHash e LSH, incluindo a análise dos *buckets* gerados e da redução do espaço de busca. Em seguida, são investigados os efeitos de diferentes limiares de similaridade de Jaccard na estrutura dos grafos, considerando métricas quantitativas e homogeneidade dos componentes conectados. Finalmente são apresentados estudos de caso sobre componentes mistos gerados em diferentes cenários de similaridade.

4.1. Pares Recuperados pelo LSH

Para avaliar o comportamento da abordagem proposta durante a geração de pares candidatos pelo LSH, o APKHash foi aplicado sobre o conjunto de dados composto por 12.000 APKs distribuídas em 24 famílias de malware (descrito na Seção 3.1). Nesse experimento, foi utilizada uma configuração do MinHash com 256 permutações, organizadas em 32 bandas com 8 linhas por banda. Ao todo, foram gerados 128.467 *buckets*, dos quais 92.479 (72%) continham apenas um item. Os 35.988 *buckets* restantes, contendo mais de um item, apresentaram em média 8 itens agrupados. Entretanto, também foram observados *buckets* anômalos, como o maior deles, que agrupou 640 itens.

A predominância de *buckets* com apenas um item era esperada no uso combinado de MinHash e LSH. Essa baixa taxa de colisão sugere que 72% das assinaturas geradas foram

suficientes para capturar padrões únicos no conjunto de malwares Android analisado. Por outro lado, os *buckets* anômalos com elevado número de colisões podem indicar a presença de trechos comuns entre diferentes APKs, representados pelos 4-gramas extraídos. Esses n-gramas provavelmente correspondem a sequências de opcodes associadas a bibliotecas amplamente utilizadas em aplicações Android. Devido ao funcionamento do MinHash e LSH não é possível reverter as assinaturas dos *buckets* para determinar quais foram os quadrigramas que deram origem a chave.

Em relação às famílias de malware, a grande maioria dos *buckets*, 122.656 (95%), contém amostras de apenas uma única família. Esse resultado era esperado, considerando que 72% dos *buckets* possuíam apenas um item. Mesmo entre os *buckets* com múltiplos itens, observa-se uma elevada taxa de colisões entre amostras da mesma família: 20.177 (56%) dos 35.988 *buckets* com colisões continham exclusivamente APKs de uma única família. Em contrapartida, 15.811 *buckets* apresentaram amostras pertencentes a mais de uma família, representando aproximadamente 12% do total de *buckets* gerados pelo LSH.

Esses *buckets* heterogêneos provavelmente refletem a existência de trechos de código compartilhados entre os malwares Android analisados. Famílias pertencentes a categorias semelhantes, como famílias de trojans, podem reutilizar bibliotecas comuns ou implementar comportamentos similares, resultando em assinaturas parcialmente similares. Ainda assim, embora existam assinaturas do LSH com convergência entre diferentes famílias, elas representam uma porcentagem pequena do total de *buckets* gerados, reforçando a capacidade do método em criar representações de famílias de malware distintas.

Outra forma de avaliar os resultados do MinHash e do LSH é por meio da redução do espaço de busca. O conjunto de dados utilizado possui 71.994.000 pares possíveis. Ao todo, os 35.988 *buckets* com mais de um item produziram 12.024.400 pares. Desse total, existem pares duplicados já que um mesmo par de APKs pode colidir em mais de uma banda. Como cada par deve ser contabilizado uma única vez essas ocorrências foram deduplicadas, resultando em 1.886.845 pares candidatos únicos. Ao considerarmos somente os pares únicos, o APKHash reduziu em aproximadamente 97,38% o número de comparações em relação à busca exaustiva

4.2. Avaliação dos limiares de similaridade

Um dos parâmetros centrais do APKHash é o limiar de similaridade de Jaccard t, utilizado para definir as arestas do grafo. Foram avaliados os valores $t \in \{0{,}5, 0{,}6, 0{,}7, 0{,}8, 0{,}9, 1{,}0\}$, abrangendo configurações progressivamente mais restritivas. Para avaliar os componentes conectados, utilizamos precisão, *recall* e F1 pareados. Um verdadeiro positivo (TP) é um par de APKs da mesma família no mesmo componente; um falso positivo (FP), um par de famílias distintas no mesmo componente; e um falso negativo (FN), um par da mesma família em componentes diferentes. Assim, $P = TP/(TP+FP)$, $R = TP/(TP+FN)$ e $F1 = 2PR/(P+R)$. As métricas pareadas, a pureza e a fração de APKs em componentes mistos consideram as 12.000 amostras, enquanto as métricas de tamanho consideram apenas componentes não unitários. A Tabela 1 resume os resultados.

Na Tabela 1, observa-se que, à medida que o limiar de Jaccard aumenta, o critério para estabelecer conexões entre os nós torna-se mais rigoroso, reduzindo gradualmente o número de nós não isolados, de arestas e o grau médio dos grafos. Em $t = 0{,}5$, o grafo apresenta 11.001 nós, 1.537.433 arestas e grau médio de 279. Já em $t = 0{,}9$, considerando

Tabela 1. Métricas estruturais e de pureza dos grafos para diferentes limiares de Jaccard.

Métrica	**Limiar de Jaccard (`t`)**					
	0.5	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
Total de Nós	11001	10530	9782	8773	7655	4805
Total de Arestas	1537433	1011265	677026	472525	344373	74204
Grau Médio	279	192	138	107	89	30
Comp. não unitários	222	365	592	837	1071	791
Tamanho Médio Comp.	49.55	28.85	16.52	10.48	7.15	6.07
Maior Comp. (# nós)	7356	6234	4451	622	464	168
Maior Comp. (%)	66%	59%	45%	7%	6%	3%
Componentes Mistos	66	93	135	158	178	126
Precisão Pareada	6.37%	7.22%	9.73%	50.79%	81.32%	84.24%
Recall Pareado	58.98%	48.37%	34.35%	15.97%	10.84%	2.09%
F1 Pareado	11.50%	12.57%	15.16%	24.30%	19.13%	4.07%
Pureza Majoritária	32.77%	43.13%	56.78%	81.83%	92.25%	95.59%
APKs em Comp. Mistos	85.43%	77.36%	66.52%	49.95%	29.07%	14.41%

a faixa não extrema dos limiares avaliados, o grafo possui 7.655 nós e 344.373 arestas. Isso corresponde a reduções de 30,4% no número de nós não isolados e de 77,6% no total de arestas. O grau médio também diminui 68,1%, passando de 279 para 89 conexões por nó. Esses resultados indicam que limiares mais elevados produzem grafos mais esparsos e seletivos, preservando relações de similaridade estrutural mais fortes entre as APKs.

À medida que o limiar de similaridade aumenta o número de nós, arestas e o grau médio dos grafos seguem uma tendência de redução, enquanto os componentes conectados tornam-se menores e mais numerosos. Em `t=0.5`, o grafo possui 222 componentes conectados, enquanto em `t=0.9` esse número aumenta para 1071 componentes. Apesar do crescimento na quantidade total de componentes, cada componente passa a agrupar menos nós, dado que o tamanho médio dos componentes reduz de 49.55 para 7.15 nós. De forma semelhante, o maior componente conectado também diminui significativamente conforme o limiar se torna mais restritivo, indicando uma fragmentação progressiva da estrutura do grafo. Além disso, embora a quantidade absoluta de componentes mistos aumente com limiares mais elevados, sua proporção em relação ao total de componentes apresenta uma redução contínua ao longo dos experimentos. Esse comportamento sugere que componentes contendo APKs de famílias distintas tornam-se relativamente menos frequentes em cenários mais restritivos. Dessa forma, o aumento do limiar de Jaccard resulta em grafos compostos por componentes menores e mais homogêneos, indicando um aumento na precisão do agrupamento de exemplares pertencentes à mesma família.

Para analisar o cenário mais restritivo, avaliamos $t = 1{,}0$, no qual uma aresta é criada apenas entre APKs com conjuntos idênticos de 4-gramas. O grafo resultante contém 4.805 nós e 74.204 arestas, e seu maior componente reúne 168 APKs. Ainda assim, 126 dos 791 componentes são heterogêneos, indicando que representações idênticas também ocorrem entre rótulos distintos. Esse resultado pode decorrer de reutilização de código,

bibliotecas ou empacotadores compartilhados, granularidade dos rótulos ou limitações da representação adotada, não constituindo, isoladamente, evidência de erro de rotulação.

Além das métricas estruturais, a Tabela 1 apresenta métricas pareadas dos componentes conectados. Com o aumento do limiar, a precisão pareada cresce, enquanto o *recall* diminui devido à maior fragmentação das famílias. O maior F1 foi obtido em $t = 0{,}8$, com 24,30%, indicando o melhor equilíbrio observado entre precisão e *recall*. À medida que o limiar de similaridade aumenta, os agrupamentos tornam-se mais homogêneos: a precisão por família cresce de 38,03% para 84,29%, a pureza majoritária aumenta de 32,77% para 92,25% e o percentual de APKs em componentes mistos cai de 85,43% para 29,07%. Em contrapartida, o recall se deteriora gradativamente, atingindo 2,09% em `t=1.0`, comportamento também refletido na redução do F1-Score. Em conjunto com o aumento no número de componentes conectados e a diminuição de seus tamanhos, esses resultados indicam maior fragmentação entre as famílias. Assim, as APKs passam a ocupar componentes menores e mais isolados porém com mais itens pertencentes a mesma família.

Os resultados observados nos experimentos relacionados ao limiar de similaridade de Jaccard mostram que a presença de componentes mistos persiste mesmo em limiares mais restritivos. Esse comportamento reforça a hipótese de que diferentes famílias de malware Android compartilham características estruturais semelhantes, seja pelo reaproveitamento de bibliotecas e trechos de código, seja pela implementação de comportamentos parecidos. Outra possibilidade é a existência de inconsistências nas rotulações fornecidas pelos mecanismos antivírus. Como consequência, APKs pertencentes a famílias distintas podem apresentar assinaturas estruturalmente similares quando representadas pelo APKHash.

5. Estudos de Caso: Componentes Conectados

Para avaliar o APKHash e investigar possíveis ruídos de rotulação, analisamos os componentes conectados nos limiares $t = 0{,}8$ e $t = 1{,}0$. O valor $t = 0{,}8$ apresentou o maior F1 pareado, representando o melhor equilíbrio observado entre precisão e *recall*, além de preservar conexões suficientes para uma análise estrutural mais próxima de um cenário prático. Já $t = 1{,}0$ representa o cenário mais restritivo, no qual apenas APKs com conjuntos idênticos de 4-gramas são conectadas. A permanência de famílias distintas nesses componentes permite investigar indícios de reutilização estrutural, compartilhamento de código ou inconsistências nos rótulos.

5.1. Componentes em t=0.8

Para analisar o cenário `t=0.80`, calculamos métricas considerando os componentes homogêneos e heterogêneos gerados pelo APKHash. A Tabela 2 sumariza essas métricas. O APKHash produziu 837 componentes conectados que agrupam 8.773 APKs, correspondendo a aproximadamente 73% de todo o conjunto de dados avaliado. As APKs restantes não formaram grupos com outras amostras e, portanto, ficaram fora dessa representação, já que o grafo exige ao menos uma aresta considerando o limiar de similaridade de Jaccard maior ou igual a 0.80. Dos componentes conectados gerados, 679 (81%) são homogêneos e os 158 restantes são heterogêneos. Embora a maioria dos componentes seja composta por APKs de uma única família, apenas 31% das APKs do conjunto estão presentes nesses componentes.

Os componentes homogêneos tendem a ser significativamente menores que os heterogêneos. Enquanto os componentes homogêneos apresentam tamanho médio de aproximadamente quatro nós, os componentes mistos possuem média de cerca de 38 nós. Além disso, os componentes heterogêneos apresentam grau médio maior, com aproximadamente 21 conexões por nó, enquanto os homogêneos possuem média de apenas três conexões. Esse comportamento sugere que os componentes mistos concentram APKs altamente conectadas, formando regiões densas do grafo nas quais múltiplas famílias compartilham características estruturais ou comportamentais semelhantes.

Apesar da presença de diferentes famílias nos componentes heterogêneos, a maior parte das conexões ainda ocorre entre APKs da mesma família. Das 445 mil arestas presentes nesses componentes, aproximadamente 343 mil (77%) correspondem a conexões intra-família, enquanto apenas 23% representam conexões entre famílias distintas. Esse resultado indica que, mesmo quando o APKHash produz componentes mistos, ainda há uma forte preservação de similaridade semântica entre as amostras agrupadas. As conexões inter-família podem refletir diferentes cenários, como reutilização de código, compartilhamento de bibliotecas e empacotadores, similaridade comportamental entre famílias distintas ou até inconsistências nas *labels* do conjunto de dados. Além disso, a presença de componentes heterogêneos grandes e densamente conectados sugere que o limiar de Jaccard igual a 0.80 ainda permite a formação de agrupamentos amplos, conectando famílias que apresentam padrões estruturais ou comportamentais semelhantes.

Tabela 2. Métricas dos componentes homogêneos e heterogêneos para o grafo com limiar de Jaccard 0.8

Métrica	Homogêneos	Heterogêneos
Número de componentes	679	158
Total de APKs	2.779	5.994
Percentual de APKs	31.68%	68.32%
Média de tamanho dos componentes	4.09 ± 8.42	37.94 ± 100.79
Grau médio	2.94 ± 8.06	21.29 ± 62.67
Número médio de famílias	1.00	2.89
Total de arestas	26.780	445.745
Arestas intra-família	26.780 (100%)	342.646 (76.87%)
Arestas inter-família	0 (0%)	103.099 (23.13%)

5.1.1. Análise (maiores componentes conectados)

Para complementar a análise da seção anterior, considerando o limiar `t= 0.8`, foram selecionados os cinco maiores componentes conectados homogêneos e heterogêneos para compreender de forma mais profunda como o APKHash está agrupando as famílias de *malware* e de observar mais de perto a estrutura formadas. A partir dos subconjuntos selecionados, gerou-se as visualizações dos grafos, que são apresentadas nas Figuras 1 e 2. Além disso, as métricas quantitativas relacionando esses componentes as famílias foram calculadas (Tabela 3).

Ao observar a Figura 1, é possível identificar grafos densos e altamente interconectados. Todos os grafos gerados, com exceção da família *mailbomb*, seguem esse

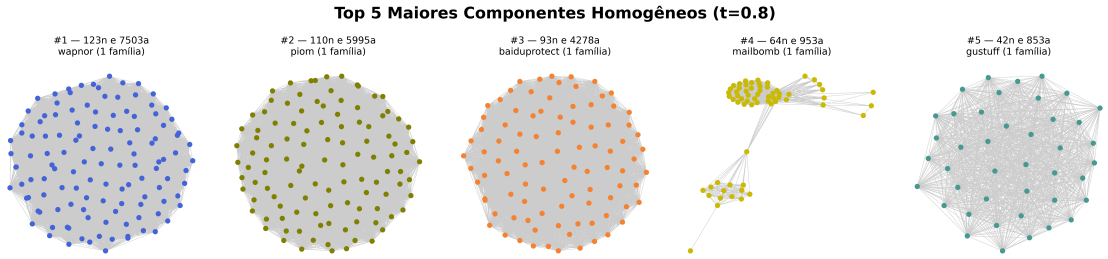

Figura 1. Maiores componentes conectados homogêneos coloridos por família utilizando t=0.8 para similaridade de Jaccard

Tabela 3. Estatísticas dos maiores componentes homogêneos e heterogêneos dos grafos de famílias.

Componentes Homogêneos (top 5 famílias)						
Família	**Nós**	**Arestas/Nó**	**Grau Médio**	**Densidade**		
wapnor	123	61	122	1.000		
piom	110	54	109	1.000		
baiduprotect	93	46	92	1.000		
mailbomb	64	14	29	0.473		
gustuff	42	20	40	0.991		
Componentes Heterogêneos (top 5 famílias)						
Família	**Nós**	**Arestas/Nó**	**Grau Médio**	**Densidade**	**% Família Maj.**	**Nº Famílias**
fakeapp	622	11	22	0.035	18.6	16
packaging*	592	196	393	0.666	63.7	7
cellspy	499	215	431	0.867	96.4	6
scamapp	438	197	394	0.903	90.9	5
gustuff	400	19	39	0.099	45.0	6

padrão de forte conectividade, apresentando em média 39 conexões por APK. Essa elevada conectividade também é refletida na métrica de densidade, na qual todas as famílias, exceto *mailbomb*, apresentam valor igual a 1.0. Esses resultados indicam que, para essas famílias, há uma assinatura bem definida e comportamentos estruturais consistentes.

Para a família *mailbomb*, na Figura 1, gráfico #4, observa-se uma densidade menor em comparação às demais famílias homogêneas. Uma análise do grafo revela a presença de um nó mais distante do *cluster* principal, que se conecta a outras APKs de forma mais dispersa. Esse nó atua como um ponto de ponte entre subestruturas, sugerindo a existência de uma possível variante do *malware*. A partir dessa variante inicial, novos nós passam a se conectar de forma progressivamente mais distribuída, afastando-se do núcleo mais denso do grafo. Nesse contexto, a representação fornecida pelo APKHash permite não apenas identificar famílias bem definidas, mas também evidenciar possíveis variantes dentro de uma mesma família de *malware*.

A Figura 2 apresenta a visualização dos maiores componentes heterogêneos gerados

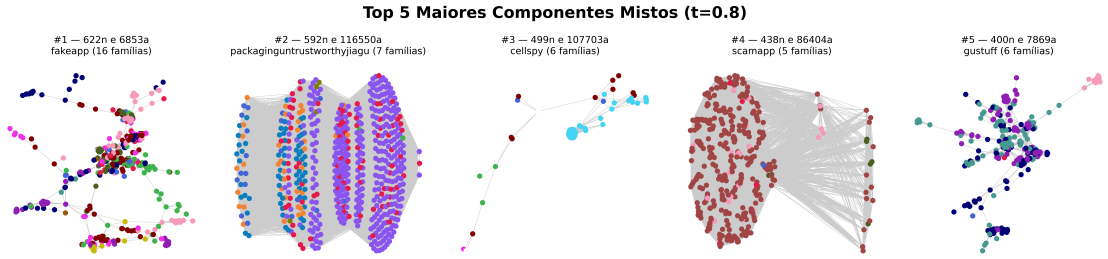

Figura 2. Maiores componentes conectados heterogêneos coloridos por família utilizando t=0.8 para similaridade de Jaccard

com limiar `t=0.8`. Observa-se uma variação significativa tanto na estrutura quanto na densidade dos grafos. Em particular, os componentes associados às famílias *fakeapp* e *gustuff* apresentam estruturas mais esparsas, com menor número de conexões entre as APKs agrupadas. O componente associado à família *fakeapp* se destaca por agrupar mais de 17 famílias distintas, sendo o mais diverso entre os componentes heterogêneos. Essa elevada diversidade se reflete diretamente em sua baixa densidade, que é a menor entre todos os componentes analisados.

Embora o componente de *gustuff* apresente um número menor de famílias em comparação ao de *fakeapp*, sua densidade permanece igualmente baixa. Esse comportamento está relacionado à baixa predominância da família majoritária dentro do componente. Para *fakeapp*, a família dominante representa apenas 18.6% das amostras, enquanto em *gustuff* essa proporção é de 45%, ainda assim insuficiente para formar uma estrutura densamente conectada, uma vez que menos da metade das APKs pertence à mesma família.

Dessa forma, as assinaturas extraídas para as APKs rotuladas como *fakeapp* e *gustuff* apresentam um comportamento mais genérico, indicando sobreposição de características com múltiplas famílias de *malware*. Esse resultado sugere que tais rótulos podem representar agrupamentos amplos de comportamentos maliciosos, em vez de famílias estritamente bem definidas. Nesse contexto, é importante destacar que sistemas de detecção baseados em assinaturas podem, em casos de baixa confiança ou alta ambiguidade, atribuir rótulos genéricos que englobam diferentes padrões de comportamento malicioso. Assim, a análise dos componentes heterogêneos com baixa conectividade entre as amostras permite identificar casos de generalização de assinatura.

Os grafos gerados para as famílias *packaginguntrustworthyjiagu*, *cellspy* e *scamapp* apresentam regiões altamente densas, com forte conectividade entre as APKs. Esses componentes construídos com o APKHash exibem os maiores percentuais de dominância da família majoritária. Em particular, o componente associado à família *cellspy* abrange seis famílias distintas, porém 96.4% das APKs pertencem ao rótulo *cellspy*. Em termos absolutos, das 499 APKs agrupadas, 479 são rotuladas como *cellspy*, enquanto apenas 20 pertencem às demais cinco famílias presentes no componente.

Devido a essa forte predominância, é possível levantar hipóteses acerca da confiabilidade dos rótulos atribuídos às amostras minoritárias. Essas APKs apresentam alta similaridade com a família *cellspy*, o que pode indicar que sejam variantes do mesmo *malware*, possivelmente rotuladas incorretamente ou classificadas sob famílias com assinaturas menos bem definidas.

Outro fator que pode explicar esse comportamento é a presença de nós intermediários (ou "nós ponte") no grafo de *cellspy*. Observa-se uma região altamente conectada e densa, acompanhada de algumas conexões mais periféricas que ligam subestruturas distintas. Esses nós de transição podem representar variantes do *malware* original, especialmente considerando que outras APKs rotuladas como *cellspy* estão localizadas em sua vizinhança no grafo, reforçando a hipótese de continuidade comportamental entre essas amostras.

5.2. Relacionamentos entre famílias

Para compreender melhor o comportamento dos rótulos de famílias de *malware*, realizamos um experimento com limiar de Jaccard igual a 1.0, o valor mais restritivo, no qual apenas APKs com assinatura idêntica pelo APKHash são conectados. Esperava-se que esse cenário eliminasse componentes heterogêneos, porém, como discutido na Seção 4.2, eles ainda foram observados. A partir desses resultados, geramos uma visualização das relações entre famílias, apresentada na Figura 3, na qual a espessura das arestas indica a intensidade das conexões entre elas, permitindo identificar possíveis inconsistências na rotulagem.

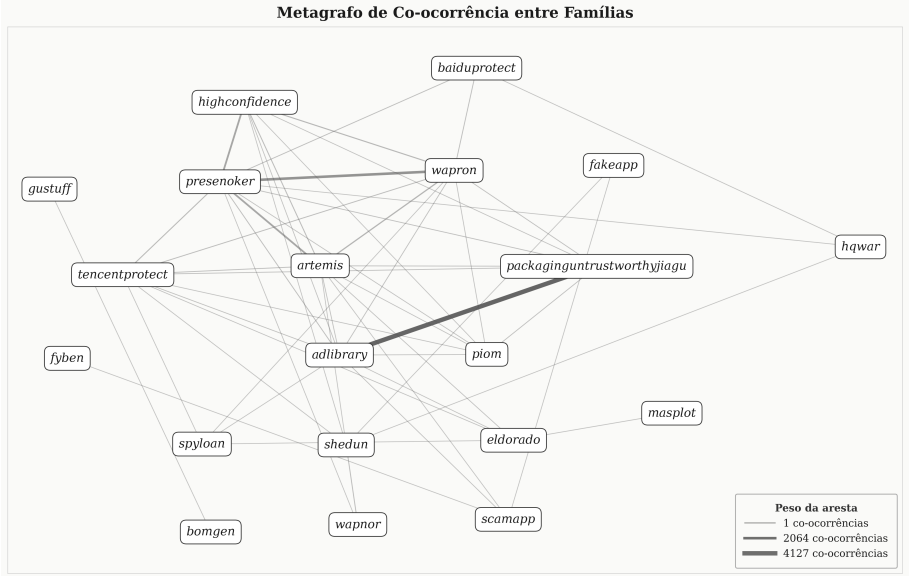

Figura 3. Co-ocorrência entre famílias. Quanto mais espessa as arestas maior é a quantidade de arestas que as famílias conectadas compartilham

Ao analisar o metagrafo apresentado na Figura 3, é possível identificar dois nós com alta centralidade, *adlibrary* e *presenoker*. Ambos apresentam forte conectividade com a maioria das demais famílias, exibindo arestas com pesos elevados, chegando a 4.127 co-ocorrências no par de maior co-ocorrência. Esse comportamento indica que essas famílias podem atuar como categorias mais genéricas no processo de rotulação de *malwares*, agregando APKs cujo padrão comportamental não é suficientemente específico para ser associado a uma família mais especializada.

Em contraste, famílias como *gustuff*, *masplot*, *spyloan* e *fyben* apresentam graus mais baixos e conexões de menor peso, o que sugere comportamentos mais bem definidos e menor sobreposição com outras classes do conjunto. Além disso, a elevada densidade de conexões envolvendo *adlibrary*, *presenoker*, *wapron* e *packaginguntrustworthyjiagu* indica a existência de uma região do espaço de comportamentos em que as fronteiras

entre famílias são pouco nítidas. Esse cenário pode influenciar diretamente o processo de rotulação dessas famílias, pois os comportamentos capturados pela assinatura são comuns a *malwares* e portanto dificultam a caracterização de comportamentos que definem essas famílias.

6. Considerações Finais e Limitações

A ferramenta APKHash para gerar grafos de similaridade e componentes conectados proposta no presente trabalho se mostrou capaz de produzir grafos com forte coesão intra-família, especialmente em limiares mais altos de similaridade, onde a maior parte das conexões ocorre entre APKs com o mesmo rótulo. Ainda assim, mesmo em configurações mais restritivas, incluindo o cenário extremo de `t=1.0`, observam-se componentes conectados mistos, o que sugere a presença de padrões estruturais compartilhados entre famílias distintas ou possíveis inconsistências nos rótulos utilizados nas bases de dados.

Apesar desses achados, a abordagem apresenta algumas limitações, tais como perda de informação estrutural no processo de extração de n-gramas e a geração de assinaturas pode introduzir perdas de informação estrutural, enquanto que a etapa de indexação via LSH pode gerar pares candidatos incorretos, afetando a construção final dos grafos. Além disso, o uso de MinHash/LSH impede a rastreabilidade direta aos n-gramas originais que originaram as similaridades, reduzindo a interpretabilidade fina dos resultados. Ainda assim, essas limitações não invalidam a aplicação do APKHash como uma ferramenta de análise estrutural em larga escala, complementar a abordagens de classificação e rotulação de *malware*, dado que mesmo em cenários restritivos, pode-se observar famílias de *malware* com exemplares muito similares em famílias distintas.

7. Conclusão

Este trabalho apresentou o APKHash, uma abordagem baseada em n-gramas de *opcodes*, MinHash/LSH e grafos de similaridade para analisar relações estruturais entre aplicações Android. A proposta foi motivada pela necessidade de investigar a consistência dos rótulos de famílias de *malware*, frequentemente definidos de forma manual e sujeitos a inconsistências entre diferentes fontes. Ao modelar APKs como grafos de similaridade, foi possível estudar tanto o agrupamento intra-família quanto a presença de conexões entre famílias distintas em larga escala. Os resultados experimentais indicam que o APKHash é capaz de capturar padrões estruturais relevantes de arquivos `.dex`, produzir grafos de similaridade que correlacionam comportamentos intra-família e inter-família. Mesmo em cenários com mais componentes conectados heterogêneos, o APKHash ainda possui a maioria das arestas conectando APKs de mesmo rótulo. Entretanto, durante os experimentos, a persistência de componentes mistos, mesmo em limiares elevados `(t=1.0)`, evidencia limitações importantes, relacionadas tanto ao compartilhamento de código entre diferentes famílias quanto a possíveis inconsistências nos rótulos fornecidos por ferramentas de detecção. Por fim, o estudo mostra que o aumento do limiar de similaridade torna os grafos mais esparsos e homogêneos, mas não elimina completamente a sobreposição entre famílias. Isso reforça a hipótese de que parte das relações observadas não se deve apenas à similaridade de código, mas também à natureza dos próprios rótulos de *malware*. Assim, o APKHash se mostra útil como uma ferramenta de análise estrutural e de diagnóstico da qualidade de *datasets* de *malware*, contribuindo para estudos futuros em curadoria de dados e construção de modelos mais robustos.

AGRADECIMENTOS

Este trabalho foi apoiado pelo Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações, bem como pelo CNPq por meio de bolsa de produtividade em pesquisa.

Referências

Allix, K., Bissyandé, T. F., Klein, J., and Traon, Y. L. (2016). AndroZoo: Collecting millions of Android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471. ACM.

Canfora, G., De Lorenzo, A., Medvet, E., Mercaldo, F., and Visaggio, C. A. (2015). Effectiveness of opcode ngrams for the detection of multi family android malware.

Crussell, J., Gibler, C., and Chen, H. (2015). Andarwin: Scalable detection of android application clones based on semantics. *IEEE Transactions on Mobile Computing*, 14(10):2007–2019.

Dai, J., Luo, M., Zhang, Y., Yang, M., and Yang, M. (2025). Apkdiffer: Accurate and scalable cross-version diffing analysis for android applications. *Proc. ACM Program. Lang.*, 9(OOPSLA2).

De Ghein, R., Abrath, B., De Sutter, B., and Coppens, B. (2022). Apkdiff: Matching android app versions based on class structure. In *Proceedings of the 2022 ACM Workshop on Research on Offensive and Defensive Techniques in the Context of Man At The End (MATE) Attacks*, Checkmate '22, page 1–12, New York, NY, USA. Association for Computing Machinery.

Frenklach, T., Cohen, D., Shabtai, A., and Puzis, R. (2021a). Android malware detection via an app similarity graph. *Computers & Security*, 109:102386.

Frenklach, T., Cohen, D., Shabtai, A., and Puzis, R. (2021b). Android malware detection via an app similarity graph. *Computers Security*, 109:102386.

Guan, Q., Huang, H., Luo, W., and Zhu, S. (2016). Semantics-based repackaging detection for mobile apps. In *Engineering Secure Software and Systems*, volume 9639 of *Lecture Notes in Computer Science*, pages 89–105. Springer.

Hadi, H. J., Khalid, A., Hussain, F. B., Ahmad, N., and Alshara, M. A. (2025). FLSH: A framework leveraging similarity hashing for android malware and variant detection. *IEEE Access*.

Hurier, M., Suarez-Tangil, G., Dash, S. K., Bissyandé, T. F., Traon, Y. L., Klein, J., and Cavallaro, L. (2017). Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 425–435. IEEE Press.

Joyce, R. J., Everett, D., Fuchs, M., Raff, E., and Holt, J. (2025). Claravy: A tool for scalable and accurate malware family labeling. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 277–286.

Júnior, C. T., Filho, D. F., Pincovscy, J., and Grégio, A. (2025). Artemis: Uma plataforma modular para execução, monitoração e investigação de aplicativos android suspeitos. In

Anais do XXV Simpósio Brasileiro de Cibersegurança, pages 147–162, Porto Alegre, RS, Brasil. SBC.

Kang, B., Yerima, S. Y., Sezer, S., and McLaughlin, K. (2016). N-gram opcode analysis for android malware detection. *International Journal on Cyber Situational Awareness*, 1(1):231–254.

Karbab, E. B., Debbabi, M., Derhab, A., and Mouheb, D. (2020). Scalable and robust unsupervised android malware fingerprinting using community-based network partitioning. *Computers & Security*, 96:101932.

Kim, H. M., Song, H. M., Seo, J. W., and Kim, H. K. (2019). ANDRO-SIMNET: Android malware family classification using social network analysis.

Lee, S., Jung, W., Kim, S., Lee, J., and Kim, J.-S. (2019). Dexofuzzy: Android malware similarity clustering method using opcode sequence. Virus Bulletin.

Li, T., Shou, P., Wan, X., Li, Q., Wang, R., Jia, C., and Xiao, Y. (2024). A fast malware detection model based on heterogeneous graph similarity search. *Computer Networks*, 254:110799.

Paulevé, L., Jégou, H., and Amsaleg, L. (2010). Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358.

Saarinen, M. J. and Aumasson, J. P. (2015). The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). RFC 7693.

Shen, L., Fang, M., and Xu, J. (2024). Ghgdroid: Global heterogeneous graph-based android malware detection. *Computers Security*, 141:103846.

Simoni, M., Saracino, A., et al. (2025). Matrix: A comprehensive graph-based framework for malware analysis and threat research. In *Proceedings of the 22nd International Conference on Security and Cryptography, SECRYPT*, pages 11–13.

Wang, L., Wang, H., Zhang, T., Xu, H., Meng, G., Gao, P., Wei, C., and Wang, Y. (2024). Android malware family labeling: Perspectives from the industry. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, page 2176–2186, New York, NY, USA. Association for Computing Machinery.

Wu, Y., Shi, J., Wang, P., Zeng, D., and Sun, C. (2023). Deepcatra: Learning flow- and graph-based behaviours for android malware detection. *IET Information Security*, 17(1):118–130.