# FbHash: A New Similarity Hashing Scheme for Digital Forensics

Timotej Knez 63..

Sebastian Mežnar 27192031

Jasmina Pegan 63170423

#### **POVZETEK**

nek povzetek

# Kategorija in opis področja

E.3 [Data encryption]

## Splošni izrazi

Hashing

### Ključne besede

Data fingerprinting, Similarity digests, Fuzzy hashing, TF-IDF, Cosine-similarity

#### 1. UVOD

Živimo v obdobju shranjevanja ogromnih količin podatkov. Pri forenzičnih preiskavah se pogosto zgodi, da je pridobljenih datotek preveč za ročno pregledovanje. Digitalni forenziki se tako soočijo s problemom avtomatizacije preiskave datotek. Možna rešitev so algoritmi, kot so ssdeep, sdhash in Fb-Hash, ki poskusijo filtrirati vnaprej znane "slabe" oziroma "dobre" datoteke. Ti algoritmi (angl. Approximate Matching algorithms) ugotavljajo delež ujemanja datotek s pomočjo (nekriptografskih) zgoščevalnih funkcij. Algoritma ssdeep in sdhash lahko preslepi aktivni napadalec, ki pametno napravi majhne spremembe na določenih mestih datoteke. Učinkovitega napada na algoritem fbhash ne poznamo.[?]

V 2. poglavju predstavimo predhodnike algoritma FbHash. V 3. poglavju podrobneje predstavimo algoritem FbHash in našo implementacijo. V 4. poglavju opišemo izvedene eksperimente in v 5. poglavju opišemo rezultate. V 6. poglavju povzamemo narejeno delo in rezultate.

#### 2. SORODNA DELA

Prvi algoritem, namenjen iskanju približnih ujemanj, je bil objavljen leta 2002 pod imenom dcfldd. Ta algoritem je razvil N. Harbour kot izboljšano verzijo ukaza dd[?]. Izboljšana

različica tega algoritma je ssdeep. Pomembnejša predhodnika algoritma FbHash sta tudi MRSH-v2 in mvHash-B. Obstaja še bbhash, ki pa je časovno potraten in ga ne bomo podrobneje opisali.

#### 2.1 ssdeep

Algoritem ssdeep je implementacija kontekstno sprožene kosovno zgoščevalne funkcije (angl. Context Triggered Piecewise Hash, CTPH), ki jo je predstavil J. Kornblum septembra 2006 v članku [?]. Algoritem temelji na detektorju neeželene elektronske pošte spamsum, ki lahko zazna sporočila, ki so podobna znanim neželenim sporočilom.

CTPH uporablja zgoščevanje po kosih (angl. *piecewise hashing*), kar pomeni, da se zgoščena vrednost izračuna na posameznih kosih fiksne dolžine. Za razliko od dcfldd, algoritem CTPH uporabi poljubno zgoščevalno funkcijo.

Drugi princip, ki ga uporablja CTPH, je zgoščevalna funkcija z drsečim oknom (angl.  $rolling\ hash$ ), ki preslika zadnjih k zlogov (bajtov) v psevdonaključno vrednost. Vsakega naslednika je tako možno hitro izračunati iz predhodno izračunane vrednosti. Pri tem je uporabljena zgoščevalna funkcija FNV.

Postopek CTPH se začne z izračunom zgoščenih vrednosti z drsečim oknom. Ob določeni sprožilni zgoščeni vrednosti (angl. trigger value) se vzporedno s tem sproži še algoritem zgoščevanja po kosih. Ob ponovni pojavitvi sprožilne vrednosti se dotlej zbrane vrednosti druge zgoščevalne funkcije zapišejo v končni prstni odtis. Tako se ob lokalni spremembi v datoteki sprememba pozna le lokalno tudi v prstnem odtisu.

Sledi primerjava prstnih odtisov datotek, ki temelji na uteženi Levenstheinovi razdalji (angl. *edit distance*), ki je nato še skalirana in obrnjena, da predstavlja 0 povsem različna prstna odtisa.

Algoritem ssdeep, ki je implementacija CTPH, se izkaže pri primerjavi podobnih besedilnih datotek in dokumentov [?]. Po drugi strani pa lahko aktivni napadalec popravi "slabe" datoteke na tak način, da se izognejo črni listi [?]. Ker je prstni odtis fiksne dolžine, je algoritem primeren le za relativno majhne datoteke podobnih velikosti.

#### 2.2 sdhash

Algoritem sdhash je opisal V. Roussev januarja 2010 v članku [?]. slab priklic. Glavna prednost tega algoritma pred predhodnimi je, da izbere statistično manj verjetne dele datotek kot izhodišče za računanje prstnega odtisa.

Postopek se začne z iskanjem statistično najmanj verjetnih delov datoteke. Izračuna se entropija skupin po k zlogov datoteke. Nato se izračuna rank vsake skupine glede na nsosednjih skupin. Izbrane so skupine, ki imajo rank večji ali enak postavljeni meji.

Sledi filtriranje skupin k zlogov, ki niso bistvene, povzročajo pa lažno pozitivne rezultate. Ocenili so, da je dobro zavreči skupine z oceno entropije pod 100 ali nad 990, ker so takšne skupine pogoste v datotekah tipa JPEG.

Nato se generira prstni odtis datoteke kot zaporedje Bloomovih filtrov, ki so verjetnostne strukture, uporabljene za prostorsko učinkovito predstavitev množic. Algoritem sdhash preveri za vsako izbrano skupino k zlogov, ali je že v množici, predstavljeni z Bloomovimi filtri. Če skupine ni v množici, jo algoritem doda.

Nazadnje algoritem primerja prstne odtise datotek, torej zaporedje Bloomovih filtrov. Za vsak filter, ki predstavlja prvo datoteko, se izračuna maksimalna ocena podobnosti s filtri, ki predstavljajo drugo datoteko. Rezultat je povprečje tako pridobljenih ocen podobnosti.

Algoritem sdhash doseže boljša priklic in preciznost kot ssdeep [?]. A tudi ta algoritem ima več pomanjkljivosti: nekaterih datoteke ne more primerjati, primerjava datoteke same s seboj lahko vrne oceno med 50 in 100 ter prvih 15 zlogov sploh ne vpliva na končni prstni odtis. Poleg naštetega aktivni napadalec lahko spremeni "slabe" datoteke na tak način, da se izognejo črni listi oziroma "dobre" datoteke tako, da se obdržijo na beli listi [?].

#### 2.3 MRSH-v2

Oktobra 2012 sta F. Breitinger in H. Baier predstavila algoritem MRSH-v2 [?], ki se opira na predhodno razvit algoritem MRSH (angl. multi-resolution similarity hashing), ta pa temelji na algoritmu ssdeep.

Algoritem MRSH ima določene sprožilne točke -1 modb, kjer b pomeni povprečno velikost bloka. Namesto zgoščevalne funkcije z drsečim oknom uporabi polinomsko zgoščevalno funkcijo djb2, kot primitiv pa MD5. Namesto konkatenacije zgoščenih vrednosti MRSH kot prstni odtis uporabi seznam Bloomovih filtrov.

Algoritem MRSH-v2 ponovno uporabi zgoščevalno funkcijo z drsečim oknom, kot ssdeep, namesto FNV pa uporabi funkcijo zgoščevanja MD5. Za večjo hitrost in v izogib napadu z dodajanjem sprožilnih točk je dodana tudi spodnja meja za velikost skupin zlogov  $\frac{b}{4}$ .

Algoritem MRSH-v2 je po [?] časovno učinkovitejši od predhodnih algoritmov. Vključuje način za odkrivanje fragmentov in način za odkrivanje podobnih datotek. Po analizi leta 2014 [?], ki primerja ssdeep, sdhash in MRSH-v2, se v povprečju najbolje obnese sdhash, ssdeep in sdhash izkazujeta dobro preciznost, vsi trije algoritmi pa imajo relativno

#### 2.4 mvHash-B

Marca 2013 so F. Breitinger in sod. predstavili algoritem mvHash-B [?]. Ideja algoritma je, da majhne lokalne spremembe ne spremenijo končnega rezultata.

V prvem koraku se izvede večinsko glasovanje po bitih z nastavljivo mejo t. Vsakih k zlogov se tako preslika v ničle, če je število enic v zaporedju bitov manjše od t, sicer pa v

Nato se zaporedje bitov zapiše na bolj kompakten način enake zaporedne bite nadomestimo z dolžino takega niza. Če se zaporedje bitov ne začne z nekaj ničlami, bo prvo število v seznamu 0. Dobimo kodirano zaporedje dolžin nizov.

Kodirano zaporedje se razdeli v prekrivajoče se skupine fiksne dolžine. Te skupine so dodane v Bloomov filter. Nazadnje se primerja Bloomove filtre s pomočjo Hammingove razdalje, ocene pa se povpreči in odšteje od 100.

Algoritem mvHash-B naj bi bil hiter skoraj kot SHA-1, torej blizu zgornje meje učinkovitosti [?]. Vendar tudi ta algoritem ni varen pred aktivnim napadalcem, saj je možno popraviti "slabo" datoteko tako, da se izogne črni listi [?].

#### 3. ALGORITEM

Avtorji v članku [?] predstavijo algoritem FbHash-B, ki je namenjen nestisnjenim (ang. uncompressed) in FbHash-S, ki je namenjen stisnjenim (ang. compressed) datotekam. Iskanje želenih datotek poteka na sledeč način. Najprej datoteke, ki jih želimo preveriti zgostimo s pomočjo ustreznega algoritma glede na njihov tip in tako dobim bazo podatkov. Nato zgostimo še ciljne datoteke in jih primerjamo z datotekami iz baze podatkov. Tako lahko datoteke, ki imajo s ciljnimi dovolj podobnosti zavržemo (ang. blacklist) oziroma dodamo med iskane (ang. whitelist).

#### 3.1 FbHash-B

FbHash-B je bolj primeren datotekam, ki niso stisnjene. V opisu algoritma bomo za boljšo preglednost uporabili sledečo notacijo:

- Del datoteke: niz k zaporednih bajtov.
- $ch_i^D$ : del datoteke D, ki se začne na i-tem bajtu.
- Frekvenca dela oziroma  $chf_{ch_i}^D$ : število pojavitev dela  $ch_i$  v datoteki D.
- Dokumentna frekvenca dela oziroma  $df_{ch}$ : število dokumentov, ki vsebujejo del datoteke ch.
- N: število datotek v bazi podatkov.
- $RollingHash(ch_i)$ : vrednost rekurzivne zgoščevalne funkcije imenovane rolling hash na delu besedila  $ch_i$
- $chw_{ch_i}$ : utež dela besedila  $ch_i$
- $docw_{ch_i}$ : dokumentna utež dela besedila  $ch_i$
- $W_{ch_i}^D$ : ocena dela besedila  $ch_i$  v dokumentu D

Algoritem je predstavljen v treh korakih. Prvi zavzema računanje frekvence delov datotek, drugi računanje dokumentnih uteži delov datotek in tretji računanje zgostitvenih uteži

#### 3.1.1 Računanje frekvence delov datotek

V tem koraku se izračunajo frekvence delov datotek in njihove uteži. Vsaka datoteka vsebuje  $N_D-k$  delov datotek, kjer je  $N_D$  dolžina datoteke D v bajtih in k parameter. Tako ima i-ti del datoteke D obliko  $B_i^DB_{i+1}^D...B_{i+k-1}^D=ch_i^D$ , kjer je  $B_i^D$  j-ti bajt v datoteki D.

Najprej izračunamo  $RollingHash(ch_0^D)$  s formulo

$$RollingHash(ch_0^D) = (B_0^D * a^{k-1} + B_1^D * a^{k-2} + ... + B_{k-1}^D * a^0) \mod n$$

, kjer so a, n in k parametri. Zaradi rekurzivne strukture funkcije RollingHash lahko zgostitve preostalih delov izračunamo na sledeč način:

$$RollingHash(ch_{i+1}^D) = (a*RollingHash(ch_i^D) - B_i^D*a^k - B_{i+k}^D) \ mod \ n$$

Parametre a, k in n izberemo na sledeč način. Za a izberemo neko konstanto med 2 in 255. Če za zalogo vrednosti funkcije RollingHash vzamemo 64 bitna števila, lahko k izračunamo na sledeč način:

$$B_i^D * a^{k-1} \le 2^{64} - 1$$

. Ker je maksimalna vrednost bajta in parametra a enaka 255 dobimo neenačbo  $255^k \leq 2^{64}-1$  iz česar sledi, da je k manjši ali enak 7. Tako za k izberemo 7. Če želimo zagotoviti, da ne pride do trkov med različnimi deli datotek, moramo za n izbrati prašteviko, ki je večje od  $2^56=255*255^{k-1}$ .

Strukturo datoteke lahko tako predstavimo kot razpršeno tabelo, kjer je kjuč vrednost, ki jo dobimo s funkcijo RollingHash na določenem delu datoteke, vrednost pa število pojavitev pripadajočega dela datoteke (oziroma vrednosti, ki smo jo dobili s funkcijo RollingHash).

Prvi korak zaključimo tako, da izračunamo uteži za dele besedila znotraj datoteke  $(chw_{ch_i})$  s pomočjo ene izmed funkcij iz ??.

- 3.1.2 Računanje dokumentnih uteži
- 3.1.3 Računanje zgostitvenih uteži

#### 3.2 FbHash-S

FbHash-S je bolj primeren za stisnjene datoteke tipa docx, pptx, pdf, zip, ...

- 3.3 Primerjanje dokumentov
- 3.4 Uporabljene funkcije za uteževanje
- 4. NAŠI EKSPERIMENTI (NAME IN PROGRESS)
- 5. REZULTATI
- 6. ZAKLJUČEK
- 7. ZAHVALA

Mogoče zahvala avtorjem za narjeno delo al kej.