Dugine tablice

Horvat, Mirjana

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:130487

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Mirjana Horvat

DUGINE TABLICE

Diplomski rad

Voditelj rada:
izv.prof.dr.sc. Filip Najman

Zagreb, Rujan, 2018

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem svome mentoru izv.prof.dr.sc. Filipu Najmanu na prenesenom znanju, pomoći i strpljenju prilikom izrade ovog dilomskog rada.
Zahvalu dugujem i svojim roditeljima te braći koji su uvijek pronašli vremena i bili uz mene. Posebno se zahvaljujem Dejanu na strpljenju i podršci tijekom cijelog studiranja i vjeri u moj usjeh.

Sadržaj

Sadržaj **iv**

Uvod **1**

1 Sigurna komunikacija i integritet poruka **2**
1.1 Enkripcija vs. autentifikacija poruke 3
1.2 Kodovi za autentifikaciju poruke - Definicije 3
1.3 ”Replay” napad 6

2 Hash-funkcije **8**
2.1 Hash-funkcije 8
2.2 Hash-funkcije otporne na kolizije 10
2.3 Općeniti ”Birthday” napad 12
2.4 ”Birthday” napad na hash-funkcijama 13
2.5 Poboljšani ”birtday” napad 14
2.6 Merkle-Damgard transformacija 15
2.7 Hash-funkcije otporne na kolizije u praksi 18

3 Dugine tablice **20**
3.1 Originalna metoda Martina Hellmana 21
3.2 Nova struktura tablica sa boljim rezultatima 23
3.3 Unaprijed izračunati hash-lanci 24
3.4 Dugine tablice 27
3.5 Obrana protiv duginih tablica 28

Bibliografija **30**

Uvod

Kroz cijelu povijest čovječanstva postoji potreba za sigurnu razmjenu informacija. U današnje vrijeme je nezamislivo razmjenjivati podatke bez uporabe računala i interneta. Kako je internet dostupan širem broju ljudi, povjerljivi podaci trebaju što veću sigurnost. Stoga danas, više nego ikada raste potreba za razvojem tehnologija koja će zaštititi naše podatke. Znanstvena disciplina koja se bavi proučavanjem metoda za sigurnu komunikaciju je kriptografija. Osnovni zadatak kriptografije je omogućiti dvjema osobama (pošiljatelj i primatelj) komunikaciju preko nesigurnog komunikacijskog kanala (internet) na način da treća osoba (protivnik), koja može nadzirati komunikacijski kanal, ne može razumjeti njihove poruke. Posebno osjetljivi podaci su podaci tipa lozinki te podataka transakcija između banke i klijenata. Takva vrsta podataka je česta meta hakiranja te stoga postoji potreba za što većom razinom sigurnosti. Stoga ćemo se u ovom radu prvo upoznati sa metodom sigurnog spremanja takve vrste podataka, a zatim ćemo se upoznati sa metodama napada takvih podataka.
U prvom poglavlju ćemo objasniti pojmove sigurne komunikacije te integriteta podataka, odnosno razlike između enkripcije te autentifikacije podataka. Sama enkripcija ne rješava problem autentifikacije podataka, stoga ćemo se upoznati sa kodovima za autentifikaciju podataka te ”replay” napadom.
Drugo poglavlje opisuje hash-funkcije koje imaju temeljnu ulogu u suvremenoj kriptografiji. Zatim ćemo opisati posebnu klasu hash-funkcija koja ima svojstvo otpornosti na kolizije te metodu za pronalaženje kolizija, odnosno ”birtday” napad. Nakon toga opisujemo metodu za izradu hash-funkcija otpornih na kolizije koja se zove Merkle-Damgardova transformacija. Na kraju poglavlje ćemo navesti hash-funkcije otporne na kolizije koje se najčešće koriste u praksi.
Računalni sustavi koji zahtijevaju provjeru lozinke moraju sadržavati bazu podataka tih lozinki, najčešće spremljenih kao hash-vrijednosti. Stoga, u trećem poglavlju opisujemo hash-lance te dugine tablice. Na kraju poglavlja opisati ćemo obranu protiv duginih tablica, odnosno objasnit ćemo pojmove kao što su vrijednost ”soli”, istezanje ključa te jačanje ključa.
Diplomski ispit napravljen je u sklopu aktivnosti Projekta KK.01.1.1.01.0004 - Znanstveni centar izvrsnosti za kvantne i kompleksne sustave te reprezentacije Liejevih algebri.

Poglavlje 1

Sigurna komunikacija i integritet poruka

Osnovni cilj u kriptografiji je omogućiti komunikaciju preko otvorenog komunikacijskog kanala na siguran način. U mnogo slučajeva, istog ili čak većeg značaja, je integritet poruke (ili autentičnost poruke) u smislu da svaka strana može provjeriti da li je poruka doista poslana od druge strane. Na primjer, uzmimo u obzir da neki veliki lanac supermaketa pošalje email dobavljaču za kupnju 10.000 sanduka nekog soka. Po primitku takvog zahtjeva, dobavljač treba uzeti u obzir sljedeće:

- Da li je narudžba autentična? Da li je doista supermarket poslao narudžbu ili je narudžbu poslao netko drugi koji je lažirao email adresu od supermarketa?
- Čak i ako je siguran da je narudžbu poslao supermarket, treba uzeti u obzir autentičnost same poruke. Da li je narudžba točno onakva kakvu je poslao supermarket, ili je narudžba promjenjena negdje na putu od protivničke strane?

Sama narudžba nije tajna te stoga nije u pitanju privatnost, nego je problem potpuno u integritetu poruke. U globalu, ne možemo se osloniti na integritet komunikacije bez poduzimanja posebnih mjera kako bi se to osiguralo. Doista, za svaku neosiguranu online narudžbu, online bankovnu transakciju, email ili SMS ne možemo biti sigurni da dolazi od danog izvora. Nažalost, ljudi općenito uzimaju ID pozivatelja ili email adresu kao ”dokaz o podrijetlu” iako ih je relativno lako falsificirati. To otvara vrata za potencijalne protivničke napade koje ne možemo spriječiti, umjesto toga ćemo jamčiti da će svaki takav pokušaj napada biti otkriven od strane primatelja i pošiljatelja.

1.1 Enkripcija vs. autentifikacija poruke

Kako se ciljevi privatnosti i integriteta poruke razlikuju, tako se razlikuju tehnike i alati za njihovo postizanje. Nažalost, pojmovi privatnosti i integriteta poruke se često zamijenjuju i nepotrebno zapliću. Enkripcija općenito ne pruža integritet poruke, stoga se enkripcija ne bi trebala koristiti s namjerom postizanja autentifikacije poruke. Netko bi mislio da se enkripcijom odmah rješava i autentifikacija poruke, zbog nepreciznog (i netočnog) razmišljanja da se kriptiranim tekstom potpuno skriva sadržaj poruke pa protivnik ne može promijeniti kriptiranu poruku na bilo koji značajan način. Unatoč intuitivnom razlaganju, ovakvo razmišljanje je potpuno pogrešno.
Kao jednostavan primjer razmotrimo slučaj da korisnik želi prenijeti neki iznos u dolarima iz svog bankovnog računa na nečiji bankovni račun. Iznos se kriptira binarno te šalje iz jedne banke u drugu. Ako protivnik promijeni samo najmanji bit, učinak je promjena iznosa za samo 1$. Ali ako promijeni jedanaesti bit, onda se iznos mijenja za čak 1.000 $. Protivnik ne zna da li povećava ili smanjuje početni iznos. Osim toga, postojanje ovog napada nije u kontradikciji sa privatnosti enkripcijske sheme.
Dakle, enkripcija bi se trebala koristiti u kombinaciji s drugim tehnikama za postizanje autentifikacije poruke.

1.2 Kodovi za autentifikaciju poruke - Definicije

Kako smo vidjeli, enkripcija ne rješava problem autentifikacije poruke. Umjesto toga, potreban je dodatni mehanizam koji će omogućiti pošiljatelju i primatelju da znaju da li je poruka mijenjana.
Cilj autentifikacije poruke je sprečavanje protivnika da promijene poruku bez da za to ne znaju pošiljatelj i primatelj. Kao i u slučaju enkripcije, to je moguće samo ako pošiljatelj i primatelj imaju neku tajnu koju protivnik ne zna (u suprotnom protivnika ništa ne sprječava da oponaša pošiljatelja poruke).

Sintaksa koda za autentifikaciju poruka

Prije nego što formalno definiramo zaštitu koda za autentifikaciju poruka (MAC), prvo ćemo definirati što je MAC i kako se koristi. Dva korisnika koji žele komunicirati pomoću autentifikacije prvo generiraju i dijele tajni ključ k prije same komunikacije. Kada jedna strana želi poslati poruku m drugoj strani, izračunava MAC oznaku (ili jednostavno oznaku) t na temelju poruke i zajedničkog ključa te šalje poruku m zajedno s oznakom t drugoj strani. Oznaka se izračunava na temelju algoritma za generiranje oznake koji će biti dan zajedno s MAC-om. Dakle, pošiljatelj poruke m izračunava $t \leftarrow Mac_k(m)$ te šalje *(m,t)* primatelju. Nakon primanja *(m,t)*, druga strana provjerava da li je t valjana oznaka

na poruci m (s obzirom na zajednički ključ) ili ne. Provjera se vrši pomoću algoritma za autentifikaciju **Vrfy** koji uzima za unos zajednički ključ, poruku m i oznaku t te pokazuje da li je oznaka valjana. Formalno:

Definicija 1.2.1. *Kod za autentifikaciju poruka (MAC) je niz vjerojatnosno polinomijalno vremenskih algoritama (Gen, Mac, Vrfy) takvih da:*

1. ***Algoritam za generiranje ključa "Gen"*** *uzima kao ulaz sigurnosni parametar* 1^n *te izračunava ključ k takav da vrijedi* $|k| \geq n$.

2. ***Algoritam za generiranje oznake "Mac"*** *uzima kao ulaz ključ k i poruku* $m \in \{0,1\}^*$ *te izračunava oznaku t. Kako ovaj algoritam može biti nasumičan, pišemo* $t \leftarrow Mac_k(m)$.

3. ***Algoritam za autentifikaciju "Vrfy"*** *uzima kao ulaz ključ k, poruku m i oznaku t. Kao izlaz daje bit b gdje je b=1 ako je poruka autentična odnosno b=0 ako nije autentična. Bez smanjenja općenitosti uzimamo da Vrfy je determinističan te pišemo* $b := Vrfy_k(m,t)$.

Potrebno je da za svaki n i svaki ključ k, Gen(1^n*) daje izlaz te da za svaki* $m \in \{0,1\}^*$ *vrijedi* $Vrfy_k(m, Mac_k(m)) = 1$.
Ako je (Gen, Mac, Vrfy) takav da za svaki k (koji dobijemo iz Gen(1^n*)) algoritam* Mac_k *definiran samo za poruke* $m \in \{0,1\}^{l(n)}$ *(i* $Vrfy_k$ *daje izlaz 0 za svaki* $m \notin \{0,1\}^{l(n)}$*), tada kažemo da je (Gen, Mac, Vrfy) MAC konačne duljine za poruku duljine l(n).*

Gotovo sigurno $Gen(1^n)$ uzima $k \in \{0,1\}^n$ na slučajan način.
Terminologija: Vjerojatnosno polinomijalno vremenski algoritam je algoritam koji se izvodi u polinomijalnom vremenu te može koristiti slučajnost da generira ne-determinističke rezultate. Pojam vjerojatnosnog, u vjerojatnosno polinomijalno vremenskom algoritmu, znači da možemo predvidjeti određene ishode s određenom vjerojatnosti. Algoritam koji ima polinomijalno vrijeme izvršavanja je algoritam koji ima vrijeme izvršavanja složenosti $O(n^c)$ gdje je c neka konstanta.

Sigurnost koda za autentifikaciju poruka

Sada definirajmo pojam sigurnosti koda za autentifikaciju poruka. Intuitivna ideja iza definicije sigurnosti je da u polinomijalnom vremenu protivnik neće generirati važeću oznaku ili "novu" poruku koja je različita od poslane (i autentificirane) poruke.
Kako bi formalno definirali pojam sigurnosti koda za autentifikaciju poruka, prvo trebamo definirati snagu protivnika te definirati "prodor". Kao i obično, uzimamo u obzir samo

vjerojatnosno polinomijalna vremena protivnika, pa je stvarno pitanje snage protivnika zapravo način međudjelovanja protivnika na strane koje komuniciraju. Protivnik promatra komunikaciju između strana koje komuniciraju te može vidjeti sve poruke koje te strane šalju zajedno s odgovarajućim MAC oznakama. Protivnik također može utjecati na sadržaj tih poruka, bilo posredno (ako vanjske radnje protivnika utječu na poruke koje šalju strane) ili izravno. Kao primjer, razmotrite slučaj kada je protivnik zapravo osobni asistent jedne od strana i ima značajnu kontrolu nad onim porukama koje ta strana šalje.
Da bismo formalno modelirali gore navedene mogućnosti, dopuštamo protivniku da zatraži MAC oznake za sve poruke po svom izboru. Formalno, protivniku dajemo pristup *MAC* algoritmu $Mac_k(\cdot)$ te protivnik može unijeti bilo koju poruku m u algoritam. Algoritam mu zatim generira oznaku $t \leftarrow Mac_k(m)$.
"Prodorom" smatramo ako je protivnik u stanju dobiti bilo koju poruku m zajedno sa oznakom t takvu da:

1. Oznaka t je autentična oznaka za poruku m (tj. $Vrfy_k(m,t) = 1$)
2. Protivnik nije prethodno zatražio MAC oznaku za poruku m (tj. od algoritma za generiranje oznake).

Uspjeh protivnika u prvom uvjetu za "prodor" je zapravo: ako protivnik pošalje *(m,t)* jednoj od strana koje komuniciraju, onda će ta strana bili prevarena, tj. mislit će da je poruka m došla od druge strane, a ne od protivnika (jer vrijedi $Vrfy_k(m,t) = 1$). Drugi uvjet za "prodor" je potreban jer protivnik uvijek može kopirati poruku i MAC oznaku koju su prethodno slale strane koje komuniciraju. Takav napad protivnika zovemo "replay" napad te se smatra "prodorom" koda za autentifikaciju poruka. No, to ne znači da "replay" napad nije od interesa za sigurnost, što ćemo i pokazati u nastavku.
MAC koji zadovoljavaju gore navedenu razinu sigurnosti se smatra egzistencijalno neospornim pod prilagodljivim napadom odabranih poruka. "Postojeća neospornost" odnosi se na činjenicu da protivnik ne može krivotvoriti valjanu oznaku ni za jednu poruku, a "prilagodljivi napad odabranim porukama" odnosi se na činjenicu da protivnik može generirati MAC oznake za bilo koju poruku koju želi, gdje se ove poruke mogu prilagodljivo odabrati tijekom samog napada.
Uzmimo u obzir sljedeći eksperiment vezan uz kod za autentifikaciju poruka $\Pi = (Gen, Mac, Vrfy)$, protivnika $\mathcal{A}$ i vrijednosti n kao sigurnosnog parametra:

Eksperiment vezan uz kod za autentifikaciju poruka Mac-forge$_{\mathcal{A},\Pi}(n)$:

1. Proizvoljan ključ k je izračunat pomoću $Gen(1^n)$.
2. Protivniku $\mathcal{A}$ je dan ulaz 1^n i pristup algoritmu $Mac_k(\cdot)$. Protivnik na kraju predstavlja par *(m,t)*.
   Neka Q predstavlja skup svih upita koje je suprarnik $\mathcal{A}$ zatražio od algoritma $Mac_k(\cdot)$.

3. Izlaz eksperimenta je 1 ako i samo ako vrijedi:

   a) $Vrfy_k(m, t) = 1$

   b) m $\notin Q$.

Sigurnost MAC-a definiramo na način da nijedan učinkovit protivnik ne može uspjeti u gore navedenim eksperimentu uz neznačajnu vjerojatnost "prodora".

Definicija 1.2.2. *Kod za autentifikaciju poruke* $\Pi = (Gen, Mac, Vrfy)$ *je egzistencijalno nepobitan pod prilagodljivim napadom odabranih poruka, odnosno siguran ako za sve vjerojatnosne polinomijalno vremenske protivnike* $\mathcal{A}$ *postoji zanemariva funkcija* ***negl*** *takva da:*

$$Pr[Mac\text{-}forge_{\mathcal{A},\Pi}(n) = 1] \leq \boldsymbol{negl}(n) \tag{1.1}$$

1.3 "Replay" napad

Naglašavamo da gore navedena definicija i kod za autentifikaciju poruka općenito ne nude zaštitu od "replay" napada u kojima se prethodno poslane poruke (i njihova MAC oznaka) ponovno šalju na jednu od strana koje komuniciraju. Ipak, "replay" napadi su ozbiljna prijetnja. Razmotrite sljedeći scenarij: korisnik Alice šalje svoj bankovni nalog za prijenos 1.000 dolara sa svog na Bobov račun. Alice prvo izračunava MAC oznaku te ju dodaje na poruku tako da banka zna da je poruka autentična. Ako je MAC siguran, Bob neće moći presresti poruku i promijeniti iznos na 10.000 dolara (jer bi to značilo falsificiranje valjane oznake). Međutim, Boba ništa ne sprečava u presretanju Alicine poruke i slanja iste poruke banci deset puta. Ako banka prihvati svaku od tih poruka, Bob će na račun primiti 10.000 dolara umjesto 1.000 dolara.

Unatoč stvarnoj prijetnji "replay" napada, MAC ne može zaštiti od takvih napada jer definicija MAC-a (Definicija 1.2.1) ne uključuje namjeru pošiljatelja (odnosno protivnika) u algoritam za autentifikaciju (te tako svaki put valjan par *(m,t)* dan algoritmu daje izlaz 1). Umjesto toga, zaštita od "replay" napada, ako je takva zaštita uopće potrebna, je ostavljena nekoj višoj razini aplikacije. Razlog zbog kojeg je definicija MAC-a strukturirana na ovaj način je takva jer nismo spremni preuzeti bilo koju semantiku u vezi s aplikacijama koje koriste MAC-ove. Osobito, odluka o tome treba li se ponovljena poruka smatrati "valjanom" smatra se potpuno ovisnom o aplikaciji.

Dvije uobičajene tehnike za sprečavanje "replay" napada uključuju upotrebu rednih brojeva ili vremenskih pečata. Osnovna ideja prvog pristupa je da svaka poruka m dodjeljuje redni broj i, a MAC oznaka se izračunava preko ulančane poruke $i \parallel m$. Ovdje pretpostavljamo da pošiljatelj uvijek dodjeljuje jedinstveni redni broj svakoj poruci te da primatelj prati koje redne brojeve je već primio. Sada, svaka uspješna "replay" poruka m će trebati falsificirati MAC oznaku na novoj ulančanoj poruci $i' \parallel m$ gdje je i' redni broj koji još nije

bio korišten.
Nedostatak korištenja rednih brojeva je da primatelj mora pohraniti popis svih prethodnih rednih brojeva koje je primio. (Iako, u redovnoj komunikaciji, pošiljatelj može jednostavno povećati redni broj svaki put kad šalje poruku te tada primatelj mora pamtiti samo zadnji primljeni redni broj.) Kako bi olakšali postupak, ponekad se koriste vremenski pečati koji daju sličan učinak. Ovdje, pošiljatelj dodaje trenutačno vrijeme (recimo, na najbližu milisekundu) poruci umjesto rednog broja. Kada primatelj dobije poruku, provjerava je li vremenski pečat unutar nekog prihvatljivog razmaka od trenutnog vremena. Ova metoda također ima nedostatke, uključujući potrebu da pošiljatelj i primatelj održavaju sinkronizirane satove te mogućnosti ”replay” napada sve dok je dovoljno brzo izveden.

Poglavlje 2

Hash-funkcije

2.1 Hash-funkcije

Jednu od temeljnih uloga u suvremenoj kriptografiji imaju kriptografske hash-funkcije, često neformalno nazvane jednosmjerne hash-funkcije. Pojednostavljena definicija za našu diskusiju:

Definicija 2.1.1. *Hash-funkcija je računalno učinkovita funkcija mapiranja binarnih nizova proizvoljne duljine do binarnih nizova fiksne duljine, nazvanih hash-vrijednosti.*

Za hash-funkciju koja za izlaz daje n-bitne hash-vrijednosti (npr., $n = 128$ ili 160) te ima poželjna svojstva vrijedi da je vjerojatnost da se nasumično odabrani niz mapira u određenu n-bitnu hash-vrijednost (slika) jednaka 2^{-n}. Osnovna ideja je da hash-vrijednost služi kao kompaktan predstavnik ulaznog niza. Za kriptografsku upotrebu, hash-funkcija h je obično odabrana tako da je računalno nemoguće pronaći dva različita ulaza koja imaju istu hash-vrijednost (tj. dva različita ulaza x i y takva da $h(x) = h(y)$), te da je za danu specifičnu hash-vrijednost y računalno nemoguće pronaći ulazni x (praslika) tako da vrijedi $h(x) = y$. Najčešće kriptografske upotrebe hash-funkcija su u digitalnim potpisima i za integritet podataka. Kod digitalnih potpisa, dugačka poruka se obično hashira (pomoću javno dostupne hash-funkcije) te se samo hash-vrijednost potpisuje. Zatim, osoba koja prima poruku hashira primljenu poruku i provjerava je li primljeni potpis točan za tu hash-vrijednost. To štedi vrijeme i prostor u odnosu na potpisivanje poruke izravno, što obično uključuje cijepanje poruke u blokove odgovarajućih veličina i potpisivanje svakog bloka pojedinačno. Napominjemo da je nemogućnost pronalaženja dvije poruke s istom hash-vrijednošću sigurnosni zahtjev, jer inače, potpis na jednoj poruci hash-vrijednosti bi bio isti kao i na drugoj, što bi potpisniku omogućilo da potpiše jednu poruku te da kasnije tvrdi da je potpisao drugu poruku.
Hash-funkcije se mogu koristiti za integritet podataka kako slijedi. Hash-vrijednost koja

odgovara određenom ulazu izračunava se u određenom trenutku. Integritet ove hash-vrijednosti je zaštićena na neki način. U kasnijem trenutku, kako bi se potvrdilo da ulazni podaci nisu promijenjeni, hash-vrijednost se ponovno izračunava pomoću dostupnog ulaza te se uspoređuje s izvornom hash-vrijednošću. Neke od primjena hash-funkcija za integritet podataka uključuju zaštitu od virusa i distribuciju softvera.
Treća primjena hash-funkcija je njihova upotreba u protokolima koji uključuju apriori obveze, uključujući neke sheme digitalnog potpisa i identifikacijske protokole.
Hash-funkcije kao što su gore opisane, obično su javno poznate i ne uključuju tajne ključeve. Kada se koriste za otkrivanje je li poruka izmijenjena, one se nazivaju kodovi za detekciju modifikacija (eng. ”Modification detection codes” ili MDC). Zato su korisne hash-funkcije koje uključuju tajni ključ te pružaju autentifikaciju podrijetla podataka kao i integritet podataka. One se nazivaju kodovi za autentifikaciju poruka (eng. ”Message authentication codes” ili MAC).

2.2 Hash-funkcije otporne na kolizije

Općenito, hash-funkcije su funkcije koje uzimaju string proizvoljne duljine te ga "sažmu" u kraći string. Klasična upotreba hash-funkcija je u strukturama podataka gdje se koriste kako bi se postiglo $O(1)$ vrijeme umetanja i traženja za pohranu skupa elemenata. Specijalno, ako je slika hash-funkcije H veličine N, onda je inicijalizirana tablica duljine N. Tada je element x pohranjen u ćeliji $H(x)$ tablice. Kako bi ponovno dobili x, dovoljno je izračunati $H(x)$ i pogledati element koji se nalazi u ćeliji $H(x)$. "Dobra" hash-funkcija za našu svrhu je ona koja daje što manje kolizija, gdje kolizija predstavlja par različitih podataka x i x' takvih da je $H(x)=H(x')$. Primjetimo da kada se dogodi kolizija, dva elementa su spremljena u istu ćeliju. Stoga, više kolizija znači višu od željene složenosti dobivanja traženog elementa. Ukratno, poželjno je da hash-funkcija dobro raširi elemente po tablici te tako minimizira broj kolizija.
Hash-funkcije otporne na kolizije su slične po principu onima koje se koriste u strukturama podataka. Hash-funkcije otporne na kolizije su također funkcije koje uzimaju string proizvoljne duljine te ga "sažmu" u string neke fiksne duljine. Također, cilj takvih hash-funkcija je izbjegavanje kolizija. Međutim, postoje bitne razlike između standarnih hash-funkcija i hash-funkcija otpornih na kolizije. Kao prvo, želja za minimizacijom kolizija u postavkama struktura podataka postaje obavezan zahtjev u postavkama kriptografije. Također, u kontekstu strukture podataka možemo pretpostaviti da je skup podataka odabran neovisno od hash-funkcije te bez ikakve namjere da izazove kolizije. Usporedno, u kontekstu kriptografije, moramo uzeti u obzir protivnika koji će uzeti takav tip podataka koji je ovisan o hash-funkciji te ima izričitu namjeru izazivanja kolizija. To znači da su zahtjevi na kriptografske hash-funkcije stroži nego li analogni zahtjevi za hash-funkcije u strukturama podataka. Stoga su hash-funkcije otporne na kolizije teže za konstruirati.

Definicija otpornosti na kolizije

Kolizija funkcije H predstavlja par različitih ulaznih podataka x i x' takvih da je $H(x)=H(x')$, u ovom slučaju također kažemo da su x i x' u koliziji pod funkcijom H. Funkcija H je otporna na kolizije ako nije moguće da bilo koji vjerojatnosno polinomijalno vremenski algoritam nađe koliziju u H. Obično ćemo biti zainteresirani za funkcije H koje imaju beskonačnu domenu (tj. kao ulaz uzimaju stringove svih mogućih duljina) i konačnu kodomenu. U takvim slučajevima, kolizija mora postojati po Dirichletovom principu kutija (engl. Pigenonhole Principle) te je stoga zahtjev da su takve kolizije "teške" za pronaći. Nekad uzimamo funkcije H takve da su i domena i kodomena konačne. U takvom slučaju, zanimat će nas samo funkcije koje sažimaju ulazne podatke, tj. one čiji su izlazni podaci kraći nego ulazni. Otpornost na koliziju je trivijalno za postići kod funkcija kod kojih sažimanje nije obavezno. Na primjer, identiteta je trivijalna funkcija otporna na kolizije.
Formalno, bavit ćemo se familijom hash-funkcija indeksiranih po "ključu" s, tj. H će

biti funkcija koja uzima dva ulazna podatka, ključ s i string x, te kao izlaz daje string $H^s(x) := H(s, x)$. Zahtjevat ćemo da je teško pronaći koliziju za proizvoljno izabrani ključ s. Ključ s nije isti kao kriptografski ključ. Naime, svi stringovi nužno ne odgovaraju važećim ključevima (tj. H^s ne mora biti definiran za određeni s) te će stoga ključ s biti generiran algoritmom *Gen*, a ne odabran proizvoljno. Također, najveća razlika je da ključ s nije tajna. Ključ se samo koristi kako bi odredili određenu funkciju H^s od familije.

Definicija 2.2.1. *Hash-funkcija je par vjerojatnosno polinomijalnih vremenskih algoritama (Gen,H) koji zadovoljavaju:*

- *Gen je vjerojatnosno polinomijalni algoritam koji uzima kao ulaz sigurnosni parametar 1^n i kao izlaz daje ključ s. Pretpostavljamo da je 1^n implicitno dano sa s.*
- *Postoji polinom l takav da H uzima kao ulaz ključ s i string $x \in \{0,1\}^*$ te kao izlaz daje string $H^s(x) \in \{0,1\}^{l(n)}$ (gdje je n vrijednost sigurnosnog parametra implicitno danog u s).*

Ako H^s je definirano samo za ulaz $x \in \{0,1\}^{l'(n)}$ gdje $l'(n) > l(n)$, tada kažemo da (Gen,H) je hash-funkcija konačne duljine za ulaz duljine $l'(n)$.

U konačnodimenzionalnom slučaju treba vrijediti da je l' veći od l. Ovo svojstvo osigurava da je funkcija, hash-funkcija u klasičnom smislu, tj. da sažima ulazni string. U općenitom slučaju nemamo zahtjeva na l jer funkcija uzima kao ulaz sve (konačne) binarne stringove, pa stoga i stringove koji su dulji od *l(n)*. Stoga, po definiciji, također sažimaju (iako samo stringove dulje nego *l(n)*).
Sada ćemo definirati sigurnost. Prvo ćemo definirati eksperiment za hash-funkciju $\Pi = (Gen, H)$, protivnika $\mathcal{A}$ te sigurnosni parametar n:

Eksperiment za pronalazak kolizija $Hash - coll_{\mathcal{A},\Pi}(n)$:

1. *Ključ se generira pokretanjem* Gen (1^n).
2. *Protivnik $\mathcal{A}$ dobiva ključ s te generira ulaze x, x'. (Ako je Π konačnodimenzionalna hash-funkcija za ulaze duljine $l'(n)$, onda trebamo $x, x' \in \{0,1\}^{l'(n)}$.)*
3. *Eksperiment daje kao izlaz 1 ako i samo ako vrijedi $x \neq x'$ i $H^s(x) = H^s(x')$. U tom slučaju kažemo da je protivnik $\mathcal{A}$ našao koliziju.*

Definicija otpornosti na kolizije kaže da nijedan učinkovit protivnik ne može naći koliziju u gore navedenom eksperimentu, osim uz zanemarivu vjerojatnost.

Definicija 2.2.2. *Hash-funkcija $\Pi = (Gen, H)$ je otporna na kolizije ako za sve vjerojatnosno vremenske polinomijalne napade protivnika $\mathcal{A}$ postoji zanemariva funkacija* ***negl*** *takva da*

$$Pr[Hash - coll_{\mathcal{A},\Pi}(n) = 1] \leq \boldsymbol{negl}(n).$$

Terminologija: Zbog jednostavnosti, referencirati ćemo se za H, H^s i $\Pi = (Gen, H)$ sa "hash-funkcije otporne na kolizije".

Slabiji pojam sigurnosti za hash-funkcije

Otpornost na kolizije je jaki zahtjev sigurnosti te ga je teško postići. Međutim, u nekim primjenama dovoljno je uzeti slabije zahtjeve. Kada promatramo kriptografske hash-funkcije, tada uobičajeno promatramo tri razine sigurnosti:

1. *Otpornost na kolizije:* Ovo je najjači zahtjev te onaj koji smo do sada proučavali.
2. *Druga otpornost na prasliku:* Neformalno, hash-funkcija ima drugu otpornost na prasliku ako za dano s i x nije moguće u vjerojatnosno polinomijalnom vremenu protivnika pronaći $x' \neq x$ takvo da vrijedi $H^s(x) = H^s(x')$.
3. *Otpornost na prasliku:* Neformalno, hash-funkcija ima otpornost na prasliku ako za dano s i $y = H^s(x)$ (ali ne i sam x), za proizvoljan x, nije moguće u vjerojatnosno polinomijalnom vremenu protivnika pronaći x' takvi da vrijedi $H^s(x') = y$.

Svaka hash-funkcija koja je otporna na kolizije također ima otpornost na prasliku i drugu otpornost na prasliku.

2.3 Općeniti "Birthday" napad

Prije nego što konstruiramo hash-funkciju otpornu na kolizije, predstavit ćemo općeniti napad koji pronalazi kolizije u svakoj hash-funkciji (iako u vremenu koje je eksponencijalno duljini ulaznog stringa). Ovaj napad posredno daje minimalnu duljinu ulaza potrebnog da hash-funkcija potencijalno bude sigurna protiv protivnika koji vrši napad određeno vrijeme. Pretpostavimo da imamo hash-funkciju $H : \{0,1\}^* \to \{0,1\}^l$. Zbog jednostavnosti uzimat ćemo hash-funkcije koje uzimaju stringove proizvoljne duljine, iako slična varijanta napada radi za konačnodimenzionalne hash-funkcije. Također ćemo izostaviti ključ s jer napad ne ovisi o ključu.
Napad radi kako slijedi:

- Izaberemo proizvoljno q ulaza $x_1, \ldots, x_q \in \{0,1\}^{2l}$.
- Izračunamo $y_i := H(x_i)$.
- Provjerimo da li su bilo koja dva y_i jednaka.

Kolika je vjerojatnost da algoritam nađe koliziju? Očito ako je $q > 2^l$, onda je vjerojatnost jednaka 1. Međutim, mi smo zainteresirani za slučajeve kada je q manji. Teško je izračunati točnu vjerojatnost u općenitom slučaju, pa ćemo umjesto toga promatrati idealan slučaj u kojem je H proizvoljna funkcija. Dakle, za svaki i pretpostavljamo da je vrijednost $y_i = H(x_i)$ uniformno raspoređena u $\{0,1\}^l$ te neovisna o prijašnjim unosima $\{y_j\}_{j<i}$ (prisjetimo se da su svi $\{x_i\}$ različiti). Tako smo reducirali naš problem na sljedeće: ako odaberemo proizvoljne vrijednosti $y_1, \ldots, y_q \in \{0,1\}^l$ uniformno, koja je vjerojatnost da nađemo različite *i,j* takve da $y_i = y_j$?
Tako opisani algoritam za pronalazak kolizija često zovemo "birthday" napad. "Birthday" problem glasi: ako se q osoba nalazi u sobi, koja je vjerojatnost da će dvije osobe imati rođendan na isti dan? (Uz pretpostavku da su rođendani uniformno raspoređeni kroz 365 dana u godini.) To je točno analogno našem problemu: ako proizvoljan y_i predstavlja rođendan osobe i, tada imamo $y_1, \ldots, y_q \in \{1, \ldots, 365\}$ odabranih uniformno. Nadalje, odgovarajući isti rođendani predstavljaju različite *i,j* takve da $y_i = y_j$ (tj. odgovorajući rođendani odgovaraju koliziji).
Kada je $q = \Theta(2^{l/2})$, tada je vjerojatnost jednaka otprilike 1/2. U slučaju rođendana, ako u sobi ima samo 23 osobe, vjerojatnost da dvije osobe imaju rođendan na isti dan je veća od 1/2.

2.4 "Birthday" napad na hash-funkcijama

Ako je izlazna duljina stringa hash-funkcije jednaka l bitova, tada "birthday" napad pronalazi kolizije sa velikom vjerojatnosti koristeći $O(q) = O(2^{l/2})$ procjenu hash-funkcije (odnosno izračuni svih hash-vrijednosti). (Sortirajući ulaze, koliziju možemo pronaći, ako postoji, u vremenu $O(l \cdot 2^{l/2})$. Zbog jednostavnosti, pretpostavljamo da procjenu H (izračuni svih hash-vrijednosti) možemo napraviti u konačnom vremenu.).
Stoga zaključujemo, da bi hash-funkcija bila otporna na kolizije u napadu kroz vrijeme T, duljina izlaza hash-funkcije mora biti najmanje $2 \log T$ bitova. Kod razmatranja asimptotskih granica sigurnosti, nema razlike između naivnog napada koji pokušava 2^l+1 elemenata i "birthday" napad koji pokušava $2^{l/2}$ elemenata: ako vrijedi $l(n) = O(\log n)$, onda oba napada imaju polinomijalno vrijeme izvršavanja, ali ako je *l(n)* super-logaritamski onda oba napada nemaju polinomijalno vrijeme izvršavanja. Štoviše, u praksi "birtday" napadi čine veliku razliku. Kao primjer, pretpostavimo da hash-funkcija ima izlaznu duljinu stringa od 128 bita. Očito je neisplativo raditi 2^{128} koraka da bi pronašli koliziju. Međutim, raditi 2^{64} koraka je unutar područja izvedivosti (iako i dalje teško). Stoga, postojanje općenitog "birthday" napada nalaže da svaka hash-funkcija koja je otporna na kolizije u praksi mora imati izlaz dulji od 128 bita. Primjetimo da "protivnik ima dovoljno dugo vremena" je samo nužan uvjet za definiciju, ali je vrlo daleko od toga da bude dovoljan uvjet. Također naglasimo da "birthday" napad radi samo za hash-funkcije otporne na kolizije. Ne postoje

općeniti napadi na hash-funkcije za drugu otpornost na prasliku ili otpornost na prasliku koji imaju vrijeme izvršavanju bolju od 2^l.

2.5 Poboljšani "birtday" napad

Općeniti "birthday" napad ima dvije velike mane. Prvo, zahtjeva veliku količinu memorije. Drugo, sam napad daje vrlo malo kontrole oko vrijednosti kolizija. Moguće je konstruirati bolji "birthday" napad koji izbjegava ove mane.
Općeniti "birthday" napad zahtjeva da protivnik spremi svih q vrijednosti $\{y_i\}$, jer protivnik ne zna unaprijed koji par vrijednosti će izazvati koliziju. Ovo je značajan nedostatak jer je općenito memorija slabiji resurs nego vrijeme. Za ilustrativni (ali potpuno ad-hoc) primjer, usporedimo 2^{60} bajtova sa 2^{60} CPU naredbi: 2^{60} bajtova je otprilike 1 milijarda gigabajta. Ako koristimo najveće komercijalno dostupne uređaje za pohranu, koji mogu spremiti otprilike 1.000 gigabajta, onda trebamo 1 milijun takvih uređaja. Suprotno tome, 2^{60} naredbi se može izvršiti u otprilike 2 godine (uz pretpostavku da CPU izvršava 25 milijarda naredbi po sekundi, što predstavlja high-end trenutno dostupnih osobnih računala). Iako je to dugo vrijeme za čekanje, to svakako predstavlja izvedivo izračunavanje. Nadalje, izračuni ove složenosti zapravo su provedeni i prije, korištenjem velikih distribuiranih mreža.
Dakle, "birthday" napad postaje mnogo izvedljiviji ako se zahtjevi za memorijom mogu smanjiti. Zapravo, moguće je izvršiti napad tipa "birthday" napada, sa sličnom vremenskom složenosti i vjerojatnosti uspjeha kao i prije, ali koristeći samo konstantnu količinu memorije. Ideja je uzeti slučajnu početnu vrijednost x_0 te za $i > 0$ izračunati $x_i := H(x_{i-1})$ i $x_{2i} := H(H(x_{2(i-1)}))$. U svakom koraku uspoređujemo vrijednosti x_i i x_{2i} te ako su jednake onda su x_{i-1} i $H(x_{2(i-1)})$ u koliziji (osim ako se ne dogodi da budu jednaki, što se događa uz zanemarivu vjerojatnost ako nastavimo modelirati H kao slučajnu funkciju). Ključna stvar je da je ovdje potrebna samo ona memorija koja je potrebna za pohranu vrijednosti x_i i x_{2i}. Za ovakav pristup se može pokazati da daje koliziju s vjerojatnošću 1/2 u $\Theta(2^{l/2})$ korak.
Druga mana koju smo spomenuli odnosi se na nedostatak kontrole nad pronalaskom vrijednosti kolizija. Iako nije nužno pronaći "smislene" kolizije kako ne bi vrijedila formalna definicija otpornosti na kolizije, ipak je lijepo vidjeti da "birthday" napadi mogu naći i "smislene" kolizije. Pretpostavimo da protivnik Eva želi naći dvije poruke x i x' takve da vrijedi $H(x) = H(x')$ te da prva poruka x predstavlja pismo poslodavca koji objašnjava zašto je otpuštena s posla, dok druga poruka x' treba biti laskavo pismo preporuke. "Birthday" napad se samo oslanja na činjenicu da su poruke $x_1, \ldots, x_q$ različite, ali te poruke ne moraju biti proizvoljno odabrane. Dakle, možemo izvršiti napad tipa "birthday" generiranjem $q = \Theta(2^{l/2})$ poruka prve vrste i q poruka drugog tipa te zatim tražeći kolizije između poruka tih dviju vrsta. Čini se nevjerojatno da se to može učiniti za gore spomenuta pisma, no ipak, pokazuje se da je lako pisati istu rečenicu na mnogo različitih načina. Na primjer, razmotrite sljedeće:

> *Teško / nemoguće / izazovno / zahtjevno* je *zamisliti / vjerovati* da ćemo *pronaći / locirati / zaposliti* drugu *osobu / zaposlenika* koja ima slične *sposobnosti / vještine / karakteristike* kao Eva. Učinila je *izvanredan / odličan* posao.

Bitno je primijetiti da je moguće kombinirati kurzivne riječi. Dakle, rečenica može biti napisana na $4 \cdot 2 \cdot 3 \cdot 2 \cdot 3 \cdot 2 = 288$ različitih načina. Ovo je samo jedna rečenica i stoga je jednostavno napisati pismo koje se može preraditi na 2^{64} različita načina (trebate samo 64 riječi koje imaju jedan sinonim). Koristeći ovu ideju, protivnik može pripremiti $2^{l/2}$ pisma koje objašnjavaju zašto je bio otpušten i $2^{l/2}$ pisma preporuke. S velikom vjerojatnošću će naći koliziju između ta dva tipa pisma. Ovaj napad zahtijeva veliku količinu memorije i ovdje se ne može upotrebljavati verzija niske memorije opisana gore.

2.6 Merkle-Damgard transformacija

Sada predstavljamo važnu metodologiju koja se naziva Merkle-Damgardova transformacija koja se naširoko koristi u praksi za izradu hash-funkcija otpornih na kolizije. Metodologija omogućuje konverziju iz bilo koje konačnodimenzionalne hash-funkcije u punopravnu hash-funkciju (tj. koja uzima ulaz proizvoljne duljine) te pritom sačuva otpornost na kolizije (uz uvjet da je početna hash-funkcija otporna na kolizije). To znači da prilikom projektiranja hash-funkcija otpornih na kolizije možemo ograničiti našu pažnju na slučaj fiksne duljine. Ovo zauzvrat čini posao oblikovanja praktičnih hash-funkcija otpornih na kolizije puno lakšim. Osim toga što se opsežno koristi u praksi, Merkle-Damgardova transformacija je zanimljiva s teoretskog gledišta, jer podrazumijeva da je sažimanje jednim bitom jednako lako (ili teško) kao sažimanje proizvoljnim brojem bitova.
Zbog konkretnosti, uzmimo u obzir slučaj da smo dobili konačnodimenzionalnu hash-funkciju otpornu na kolizije koja sažima svoj ulaz za pola, tj. ulaz duljine $l'(n) = 2l(n)$ daje izlaz duljine *l(n)*. Označimo konačnodimenzionalnu hash-funkciju otpornu na kolizije sa *(Gen,h)* te ju upotrijebimo za izradu hash-funkcija otpornih na kolizije *(Gen,H)* koja mapira ulaze proizvoljne duljine sa izlazima duljine *l(n)*. (*Gen* će ostati nepromijenjen.)

Neka je *(Gen,h)* konačnodimenzionalna hash-funkcija otporna na kolizije koja za ulaze duljine *2l(n)* daje izlaz duljine *l(n)*. Konstruiramo beskonačnodimenzionalnu hash-funkciju otpornu na kolizije *(Gen,H)* na sljedeći način:

- Gen: ostaje nepromijenjen
- H: ako je ulaz ključ s te string $x \in \{0,1\}^*$ duljine $L < 2^{l(n)}$, onda postavi $l = l(n)$. Nadalje:
  1. Postavi $B := \lceil \frac{L}{l} \rceil$ (tj. broj blokova u x). Nadopuni x sa nulama tako da je duljina dijeljiva sa l. Rasčlani nadopunjen rezultat na niz l-bitnih blokova $x_1, \ldots, x_B$. Postavi $x_{B+1} := L$, gdje je L kodiran pomoću točno l bita.
  2. Postavi $z_0 := 0^l$.
  3. Za $i = 1, \ldots, B+1$, izračunamo $z_i := h^s(z_{i-1} \parallel x_i)$.
  4. Izlaz je z_{B+1}

Merkle-Damgardova transformacija

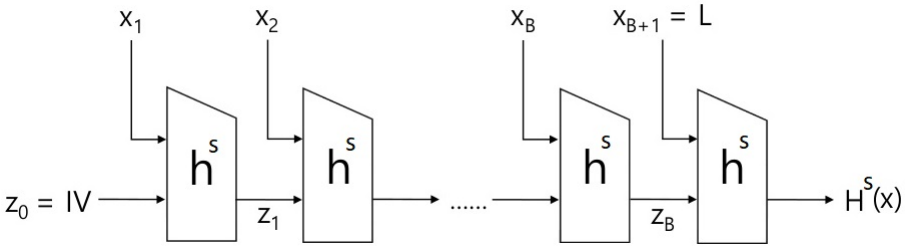
Duljinu od x ograničavamo na najviše $2^{l(n)} - 1$ kako bi se duljina L mogla rasčlaniti kao cijeli broj duljine *l(n)*.
Terminologija: Oznaka "$\parallel$" predstavlja oprerator povezivanja.

Inicijalizacija vektora

Vrijednost z_0 korištena u koraku 2. je proizvoljna i može se zamijeniti bilo kojom konstantom. Ta se vrijednost obično naziva *IV* ili *inicijalizacijski vektor*.

Sigurnost Merkle-Damgardove transformacije

Intuicija iza sigurnosti Merkle-Damgardove transformacije jest da ako su dva različita stringa x i x' u koliziji pod funkcijom H^s, tada moraju postojati različite srednje vrijednosti $z_{i-1} \parallel x_i$ i $z'_{i-1} \parallel x'_i$ u izračunu na $H^s(x)$ i $H^s(x')$, respektivno, takve da vrijedi $h^s(z_{i-1} \parallel x_i) = h^s(z'_{i-1} \parallel x'_i)$. Drugim rječima, pod funkcijom H^s može doći do kolizije, ako postoji kolizija pod početnom h^s. To ćemo pokazati u dokazu teorema 2.6.1 tako da ćemo pokazati da ako ne dođe do kolizije za h^s, tada x i x' moraju biti jednaki (suprotno početnoj pretpostavci da x i x' predstavalju koliziju pod H^s).



Slika 2.1: Merkle-Damgardova transformacija

Teorem 2.6.1. *Ako (Gen,h) je konačnodimenzionalna hash-funkcija otporna na kolizije, tada je (Gen,H) također hash-funkcija otporna na kolizije.*

Dokaz. Pokazat ćemo da za svaki s, kolizija pod funkcijom H^s daje koliziju pod funkcijom h^s.
Neka su x i x' dva različita stringa, odgovarajućih duljina L i L', takva da vrijedi $H^s(x) = H^s(x')$. Neka su $x_1, \ldots, x_B$ B-blokovi od nadopunjenog x, a $x'_1, \ldots, x'_B$ B′ -blokovi od nadopunjenog x'.
Prisjetimo se da vrijedi $x_{B+1} = L$ i $x'_{B'+1} = L'$.
Postoje dva slučaja koja trebamo razmotriti:

1. *Slučaj 1: $L \neq L'$*
   U ovom slučaju, posljednji korak izračuna $H^s(x)$ je $z_{B+1} := h^s(z_B \parallel L)$ dok je posljednji korak izračuna $H^s(x')$ $z'_{B'+1} := h^s(z'_{B'} \parallel L')$. Međutim, $L \neq L'$ pa su $z_B \parallel L$ i $z'_{B'} \parallel L'$ dva različita stringa koja su u koliziji pod funkcijom h^s.

2. *Slučaj 2: $L = L'$*
   Primjetimo da tada vrijedi $B = B'$ i $x_{B+1} = x'_{B'+1}$. Neka su z_i i z'_i srednje hash-vrijednosti od x i x' tijekom računanja $H^s(x)$ i $H^s(x')$, respektivno. Kako vrijedi $x \neq x'$ te $|x| = |x'|$, onda mora postojati barem jedan indeks i ($1 \leq i \leq B$) takav da vrijedi $x_i \neq x'_i$. Neka je $i^* \leq B+1$ najveći indeks za koji vrijedi $z_{i^*-1} \parallel x_{i^*} \neq z'_{i^*-1} \parallel x_{i'^*}$. Ako je $i^* = B + 1$, onda su $z_B \parallel x_{B+1}$ i $z'_B \parallel x'_{B+1}$ dva različita stringa koja su u koliziji pod funkcijom h^s jer

   $$h^s(z_B \parallel x_{B+1}) = z_{B+1} = H^s(x) = H^s(x') = z'_{B+1} = h^s(z'_B \parallel x'_{B+1}) \tag{2.1}$$

   Ako je $i^* \leq B$, onda maksimalnost od i^* povlači da je $z_{i^*} = z'_{i^*}$.
   Stoga, $z_B \parallel x_{B+1}$ i $z'_B \parallel x'_{B+1}$ su dva različita stringa koja su u koliziji pod funkcijom h^s.

Slijedi da bilo kakva kolizija pod hash-funkcijom H^s daje koliziju pod konačnodimenzionalnom hash-funkcijom h^s. □

2.7 Hash-funkcije otporne na kolizije u praksi

Kao u slučaju pseudoslučajnih funkcija / permutacija, konstrukcije hash-funkcije otporne na kolizije dolaze u dva oblika: dokazano sigurne konstrukcije temeljene na određenim teoretskim pretpostavkama ili visoko učinkovite konstrukcije koje su više heurističke prirode. Za sada skrenimo pozornost na drugi tip, koji uključuje one hash-funkcije koje se koriste isključivo u praksi.

Jedna važna razlika između hash-funkcija otpornih na kolizije u praksi i hash-funkcija otpornih na kolizije kako smo ih gore predstavili, je da hash-funkcije koje se koriste u praksi općenito nemaju ključ. To znači da je definirana fiksna hash-funkcija H te više nema algoritma *Gen* koji generira ključ za H. S čisto teoretskog gledišta, potrebno je uključiti ključeve u bilo koju raspravu o hash-funkcijama otpornih na kolizije jer je teško definirati smisleni pojam otpornosti na kolizije za funkcije bez ključa. Najviše što se može tvrditi o hash-funkcijama bez ključa je da nije moguće pronaći algoritam, koji se izvodi u nekom "razumnom" vremenu (recimo, 75 godina), koja pronalazi koliziju pod funkcijom H. U praksi je takva vrsta sigurnosnog jamstva dovoljna.

Čak i na pragmatičnoj razini, hash-funkcije s ključem imaju prednost: ako se ikad pronađe kolizija pod hash-funkcijom bez ključa H (recimo, pomoću brute-force pretrage), onda H više nije otporna na kolizije u bilo kojem smislenom smislu, te se mora zamijeniti. Ako je H funkcija s ključem, onda kolizija pod funkcijom H^s, koja je pronađena pomoću brute-force pretrage, ne mora nužno olakšati pronalaženje kolizije pod funkcijom $H^{s'}$ za svježe generirani ključ s'. Stoga se H može nastaviti koristiti sve dok se ključ ažurira.

Iako se ne možemo nadati dokazu otpornosti na kolizije za hash-funkcije koje se koriste u praksi (posebno zbog toga što nemaju ključ), to ne znači da ćemo potpuno ukloniti dokaze pri korištenju takvih hash-funkcija unutar neke veće konstrukcije (na primjer, konstrukcije koje konačnodimenzionalne (temeljne) hash-funkcije otporne na kolizije nadograđuju u beskonačnodimenzionalne hash-funkcije otporne na kolizije). Svi dokazi sigurnosti koji se oslanjaju na otpornost na kolizije pokazuju da, ako se uzme u obzir da se konsturkcija može "razbiti" od stane protivnika u polinomijalnom vremenu, onda se kolizija može naći u temeljnoj hash-funkciji u polinomijalnom vremenu (vidjeli smo jedan takav primjer dok smo dokazivali sigurnost Merkle-Damgardove transformacije). Ako smatramo da je u određenom vremenskom razdoblju teško pronaći koliziju pod određenom hash-funkcijom, onda to daje razumno jamstvo sigurnosti za veću konstrukciju.

Vratimo se na praktične konstrukcije i raspravi o "birthday" napadu koji smo diskutirali ranije. Takva konstrukcija je dala donju ogradu na duljinu izlaza hash-funkcije koja je potrebna kako bi se postigla određena razina sigurnosti: ako želimo da hash-funkcija bude otporna na kolizije protiv protivnika koji radi u vremenu 2^l, tada izlazna duljina hash-funkcije mora biti najmanje $2l$ bita. Dobre hash-funkcije otporne na kolizije u praksi imaju izlazni duljinu od najmanje 160 bita, što znači da bi "birthday" napad mogao potrajati 2^{80},

što je nemoguće.
Dvije popularne hash-funkcije su MD5 i SHA-1. (Kao što je objašnjeno u nastavku, zbog nedavnih napada MD5 više nije siguran te se ne smije upotrebljavati ni u jednoj aplikaciji koja zahtijeva otpornost na kolizije. Ovdje ga uključujemo jer se MD5 još uvijek koristi u naslijeđenom kodu.). I MD5 i SHA-1 najprije definiraju funkciju kompresije koja relativno malo sažima ulaze fiksne duljine (u našem smislu ova funkcija kompresije je konačnodimenzionalna hash-funkcija otporna na kolizije). Zatim se Merkle-Damgardova transformacija (ili nešto vrlo slično) primjenjuje na funkciju kompresije kako bi se dobila hash-funkcija otporna na kolizije za ulaze proizvoljne duljine. Duljina izlaza od MD5 je 128 bita, a od SHA-1 160 bita. Veća duljina izlaza od SHA-1 čini općeniti "birthday" napad težim: za MD5 "birthday" napad treba $\approx 2^{128/2} = 2^{64}$ izračuna hash-vrijednosti, dok SHA-1 za takav napad treba $\approx 2^{160/2} = 2^{80}$ izračuna hash-vrijednosti.
U 2004. godini, tim kineskih kriptoanalitičara predstavio je napad na MD5 i niz povezanih hash-funkcija. Njihova tehnika za pronalaženje kolizija daje malu kontrolu nad pronađenim kolizijama. Unatoč tome, kasnije je pokazano da se njihova metoda (i bilo koja metoda koja pronalazi "slučajnu koliziju") može koristiti za pronalaženje kolizija između, na primjer, dvije postskript datoteke te generiranja željenog sadržaja. Godinu dana kasnije, kineski tim je (teorijski) pokazao napade na SHA-1 koji bi pronašao kolizije za manje vremena nego što zahtijeva općeniti "birthday" napad. Napad na SHA-1 zahtijeva vrijeme 2^{69} koje se nalazi izvan trenutnog raspona izvedivosti. Još uvijek nije pronađena eksplicitna kolizija u SHA-1. (Što je velika razlika od napada na MD5, koji otkriva kolizije u nekoliko minuta.)
Ovi napadi potaknuli su pomak prema jačim hash-funkcijama s većim brojem izlaza koji su manje osjetljivi na poznati skup napada na MD5 i SHA-1. U tom je smislu poznata SHA-2 familija koja proširuje SHA-1 i uključuje hash-funkcije s 256 i 512-bitnim duljinama izlaza. Zbog velike količine napada postoji veliki interes za projektiranje novih hash-funkcija i razvoj novih hash standarda.

Poglavlje 3

Dugine tablice

Bilo koji računalni sustav koji zahtijeva provjeru lozinke mora sadržavati bazu podataka lozinki, bilo hashirane ili u tekstualnom obliku te stoga postoje različite metode pohrane lozinki. Budući da su tablice često meta hakiranja, pohranjivanje lozinki u tekstualnom obliku je opasno. Stoga većina baza podataka pohranjuje lozinke korisnika kao kriptografske hash-vrijednosti. U takvom sustavu, nitko (pa čak ni sustav za autentifikaciju podataka) ne može odrediti što je lozinka korisnika samo gledanjem vrijednosti pohranjene u bazi podataka. Umjesto toga, kada korisnik unese lozinku kako bi se ulogirao u sustav, lozinka se hashira te se hash-vrijednost uspoređuje sa spremljenim ulazom (koji je bio hashiran prije spremanja) za tog korisnika. Ako se hash-vrijednosti podudaraju, onda se tom korisniku odobrava pristup. Ako bi u sustav unijeli hashiranu vrijednost lozinke, sustav ne bi prepoznao tu vrijednost kao točnu vrijednost jer bi tu hash-vrijednost ponovno hashirao te dobio neku treću vrijednost. Da bi hakirali korisnikovu lozinku, poretebno je pronaći lozinku koja hashiranjem daje istu hash-vrijednost kao i korisnikova lozinka.
Dugine tablice su razvijene kako bi se izvela lozinka direktno iz hash-vrijednosti. No, napomenimo da dugine tablice nisu uvijek potrebne, jer postoje više jednostavnijih metoda preokretanja hash-vrijednosti kao što su ”brute-force” napad i ”dictionary” napad. Međutim, dugine tablice su potrebne za sustave koje koriste duge lozinke zbog poteškoća sa pohranjivanjem svih mogućih opcija te traženja u takvoj opsežnoj bazi kako bi pronašli traženu hash-vrijednost. Iako pretraživanjem hash-vrijednosti u tablici lanaca iziskuje više vremena izračunavanja, sama tablica pretraživanja može biti puno manja, tako da se mogu pohraniti i hash-vrijednosti duljih lozinki. Dugine tablice su zapravo usavršenje tehnike lanaca te pružaju rješenje problema sudara u lancima.
Dugine tablice su zapravo unaprijed izračunate tablice za preračunavanje kriptografskih hash-funkcija te obično služe za hakiranje lozinka. Tablice se obično koriste za oporavak lozinke (ili brojeve kreditnih kartica i sličnog) do određene duljine koja se sastoji od ograničenog broja znakova. To je praktičan primjer kompromisa prostora (memorije) i vre-

mena: manje vremena za obradu podataka uz više memorije nego brute-force napad (koji računa hash-vrijednost u svakom pokušaju), ali više vremena za obradu podataka uz manje memorije nego jednostavne tablice pretraživanja s jednim ulazom po hash-vrijednosti. Dugine tablice je otkrio Philippe Oechslin kao primjenu ranijeg jednostavnijeg algoritma kojeg je napisao Martin Hellman.

3.1 Originalna metoda Martina Hellmana

Za dani fiksni otvoreni tekst P_0 i odgovarajući šifrirani teks (šifrat) C_0, metoda pokušava pronaći ključ $k \in \mathbb{N}$ koji je korišten kod šifriranja otvorenog teksta pomoću šifre S. Stoga imamo:

$$C_0 = S_k(P_0). \tag{3.1}$$

Pokušavamo unaprijed generirati sve moguće šifrate tako što šifriramo otvoreni tekst sa svim N mogućim ključevima. Šifrati su organizirani u lance pri čemu se u memoriji pohranjuje samo prvi i zadnji element lanca. Pohranjivanje samo prvog i zadnjeg elementa lanca daje kompromis između memorije i vremena tako što smanjuje utrošenu memoriju po cijeni vremena kriptoanalize. Lanci se stvaraju pomoću funkcije redukcije R koja stvara ključ iz šifrata. Sam šifrat je dulji od ključa pa zato koristimo redukciju. Uzastopnom primjenom šifre S i redukcijske funkcije R možemo napraviti lance naizmjenično od ključeva i šifrata:

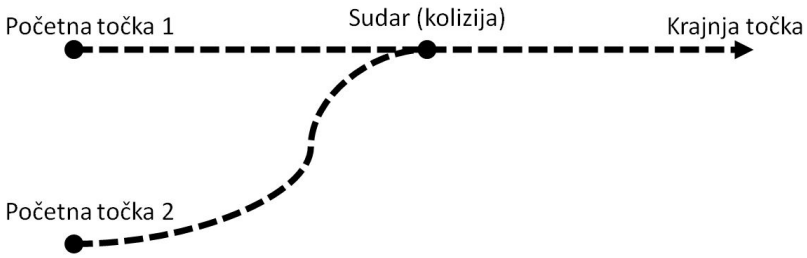
$$k_i \xrightarrow{S_{k_i}(P_0)} C_i \xrightarrow{R(C_i)} k_{i+1}. \tag{3.2}$$

Niz $R(S_k(P_0))$ zapisujemo kao $f(k)$ te on generira ključ iz kojeg se dalje generira novi ključ što vodi do lanca ključeva:

$$k_i \xrightarrow{f} k_{i+1} \xrightarrow{f} k_{i+2} \xrightarrow{f} \dots \tag{3.3}$$

Tako generiramo m lanaca duljine l te njihove prve i zadnje elemente spremimo u tablicu. Ako nam je dan šifrat C, onda možemo pokušati saznati je li ključ korišten za generiranje šifrata C među onima koji se koriste za generiranje tablice. Da bismo to učinili, prvo generiramo lanac ključeva duljine t koji počinje sa $R(C)$. Ako je C doista dobiven sa ključem za generiranje tablice, onda ćemo s vremenom generirati i ključ koji odgovara zadnjem ključu odgovarajućeg lanca. Zadnji ključ je pohranjen u memoriji zajedno s prvim ključem lanca, pa koristeći prvi ključ lanca možemo ponovno generirati čitav lanac, a osobito ključ koji dolazi neposredno prije $R(C)$. To je ključ koji je korišten kod generiranja C te zapravo i ključ koji tražimo.
Nažalost, postoji mogućnost da se lanci koji počinju s različitim ključevima "sudare" (dolazi do kolizije) i spoje. Razlog tome je činjenica da je funkcija R proizvoljna redukcija

prostora šifrata u prostor ključeva. Što je veća tablica, veća je vjerojatnost da se novi lanac spoji s prethodnim. Svako spajanje smanjuje broj različitih ključeva koji su zapravo ”pokriveni” tablicom.



Slika 3.1: Sudaranje dvaju lanaca

Vjerojatnost pronalaženja ključa korištenjem tablice sa m redaka sa ključevima duljina t je:

$$P_{table} \geq \frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \left(1 - \frac{it}{N}\right)^{j+1}. \tag{3.4}$$

Učinkovitost tablice se smanjuje kako se povećava veličina same tablice. Kako bi postigli veću vjerojatnost uspjeha, bolje je generirati više tablica koristeći različite funkcije redukcije za svaku tablicu. Vjerojatnost uspjeha korištenjem l tablica je dana sa:

$$P_{uspjeh} \geq 1 - \left(1 - \frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \left(1 - \frac{it}{N}\right)^{j+1}\right)^{l}. \tag{3.5}$$

Lanci različitih duljina tablica se mogu sudariti, ali se neće spojiti zbog korištenja različitih funkcija redukcija za različite tablice.

Lažna uzbuna

Prilikom traženja ključa u tablici, pronalaženje odgovarajuće krajnje točke (zadnji ključ) ne znači da je ključ u tablici. Doista, ključ može biti dio lanca koji ima istu krajnju točku, ali nije u tablici. U tom slučaju generiranje lanca iz spremljene početne točke ne daje ključ te takav slučaj nazivamo ”lažna uzbuna”. Lažna uzbuna se pojavljuje i kada je ključ u lancu koji je dio tablice, ali koji se spaja s drugim lancem te tablice. U tom slučaju, nekoliko početnih točaka odgovara istoj krajnjoj točki te je potrebno generirati nekoliko lanaca dok se konačno ne pronađe traženi ključ.

3.2 Nova struktura tablica sa boljim rezultatima

Glavno ograničenje originalne metode Martina Hellmana jest spajanje dva lanca prilikom sudara unutar iste tablice.
Dugine tablice koriste uzastopnu redukcijsku funkciju za svaku točku u lancu. Počinju s redukcijskom funkcijom 1, a završavaju s redukcijskom funkcijom $t-1$. Stoga, ako se dva lanca sudare, onda se oni spajaju samo ako se sudar pojavljuje na istom položaju u oba lanca. Ako se sudar ne pojavi na istom mjestu, oba lanca će nastaviti s drugačijom redukcijskom funkcijom te se neće spojiti. Ako dođe do sudara lanaca duljine t, onda će se oni spojiti s vjerojatnošću $\frac{1}{t}$. Vjerojatnost uspjeha u tablici veličine $m \times t$ jednaka je:

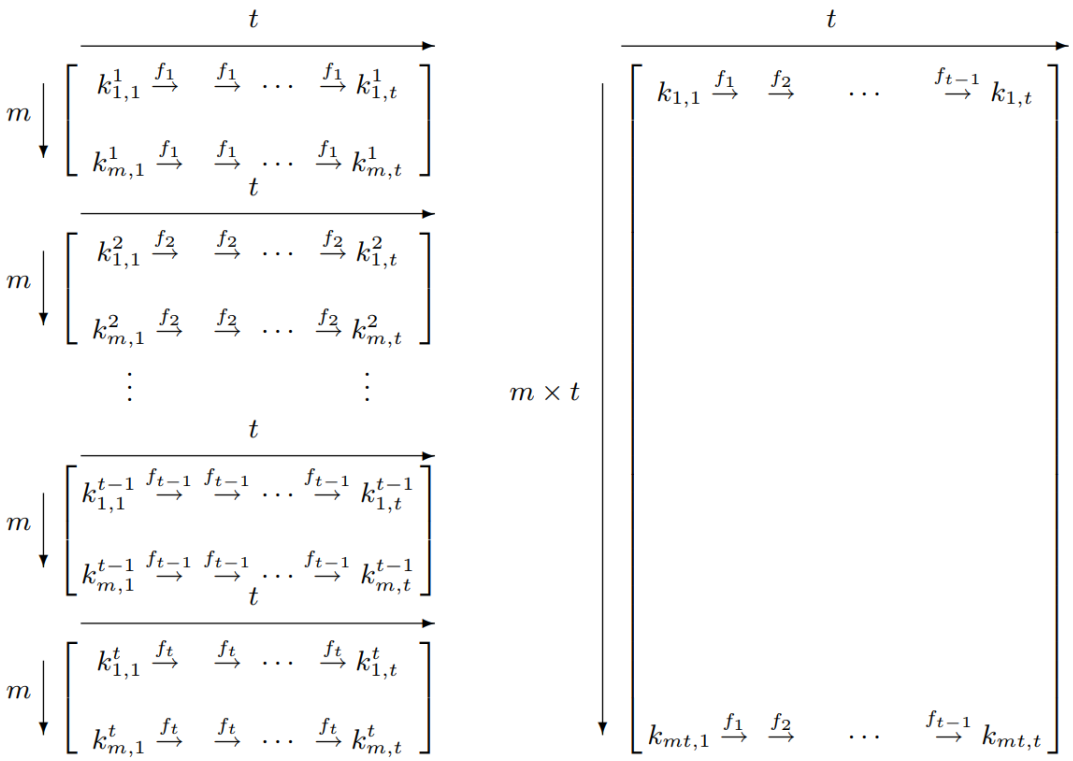
$$P_{tablica} = 1 - \prod_{i=1}^{t} \left(1 - \frac{m_i}{N}\right) \tag{3.6}$$

gdje je $m_1 = m$ i $m_{n+1} = N\left(1 - e^{-\frac{m_n}{N}}\right)$.

Zanimljivo je napomenuti da se vjerojatnost uspjeha duginih tablica može izravno usporediti s klasičnim tablicama. Doista, vjerojatnost uspjeha t klasičnih tablica veličine $m \times t$ približno je jednaka vjerojatnosti uspjeha jedne dugine tablice veličine $mt \times t$. U oba slučaja, tablice pokrivaju mt^2 ključeva sa t različitih redukcijskih funkcija. Za svaku točku sudara unutar skupa mt ključeva (pojedina klasična tablica ili stupac u duginoj tablici) rezultira spajanjem, dok sudari s preostalim ključevima ne rezultiraju spajanjem. Relacija između t tablica veličina $m \times t$ te dugine tablice je prikazana na slici 3.2.

Traženje ključa u duginoj tablici:

Prvo primjenimo R_{n-1} na šifrat te pogledamo je li rezultat u tablici krajnjih točaka. Ako pronađemo krajnju točku, onda znamo kako ponovno generirati lanac koristeći odgovarajuću početnu točku. Ako ne pronađemo krajnju točku, onda pokušamo pronaći krajnju točku primjenom R_{n-2}, f_{n-1} kako bi vidjeli da li se ključ nalazi u predzadnjem stupcu tablice. Ako ponovno ne pronađemo krajnju točku, onda primjenimo R_{n-3}, f_{n-2} , f_{n-1} te nastavljamo dalje tako dugo dok ne pronađemo krajnju točku. Stoga, ukupan broj izračuna koje moramo napraviti je jednak $\frac{t(t-1)}{2}$. To je duplo manje nego kao kod klasične metode. Doista, trebamo t^2 izračuna kako bi pretražili odgovarajućih t tablica veličine $m \times t$.



Slika 3.2: Na lijevoj strani su t klasičnih tablica veličine $m \times t$, a na desnoj strani jedna dugina tablica veličine $mt \times t$. U oba slučaja spajanja se mogu dogoditi unutar mt ključeva, a sudar se može dogoditi u preostalim $m(t$ - 1) ključeva. Potrebno je duplo više operacija za traženje ključa u duginoj tablici nego u t klasičnih tablica.

3.3 Unaprijed izračunati hash-lanci

Pretpostavimo da imamo hash-funkciju H te konačan skup lozinki P. Cilj je unaprijed generirati strukturu podataka koja za dani izlaz h hash-funkcije može locirati element $p \in P$ takav da vrijedi $H(p) = h$ ili ustanoviti da ne postoji takav p. Najjednostavniji način je izračunavanje $H(p)$ za svaki $p \in P$, no za spremanje takve tablice potrebno je $O(|P|n)$ bitova memorije, gdje je n duljina izlaza hash-funkcije H, što nije moguće za velike $|P|$.
Jedna od tehnika za smanjenje ovog zahtjeva na memoriju su hash-lanci. Ideja je definirati funkciju redukcije R koja mapira hash-vrijednosti natrag u vrijednosti iz P. Napominjemo da funkcija redukcije nije inverzna hash-funkcija. Naizmjeničnim korištenjem hash-funkcije i funkcije redukcije, formiraju se lanci koji su popunjeni naizmjenično s lozinkama i hash-vrijednostima. Na primjer, ako je P skup svih lozinki sa točno 6 malih

slova, a duljina hash-vrijednosti je 32 bita, onda bi lanac mogao izgledati ovako:

$$mhmhmh \underset{H}{\rightarrow} 08ca0280 \underset{R}{\rightarrow} nayxed \underset{H}{\rightarrow} 08f9028a \underset{R}{\rightarrow} ntgvkh$$

Jedini uvjet na funkciju redukcije je da vraća "otvoreni tekst" određene duljine.
Kako bi generirali tablicu, odabiremo slučajni skup inicijalnih lozinka iz skupa P te izračunavamo lance određene fiksne duljine k za svaku od tih inicijalnih lozinka te spremamo samo prvu i zadnju lozinku u svakom lancu. Prvu lozinku zovemo početna točka, a zadnju lozinku zovemo krajnja točka. U gornjem primjeru, "mhmhmh" je početna točka, a "ntgvkh" je krajnja točka te nijedna od ostalih lozinki (i hash-vrijednosti) neće biti pohranjena.
Sada, s obzirom na hash-vrijednost h koju želimo pretvoriti (pronaći odgovarajuću lozinku), generiramo lanac koji počinje s h tako što primjenjujemo naizmjenično R pa zatim H. Ako u bilo kojem trenutku dobijemo vrijednost koja odgovara jednoj od krajnjih točaka u tablici, onda je ta točka početna točka koju koristimo za ponovno generiranje lanca. Postoji dobra šansa da će taj lanac sadržavati hash-vrijednost h te ako sadrži, onda je vrijednost koja je neposredno prije zapravo tražena lozinka p.
Na primjer, ako smo dobili hash-vrijednost "$08f9028a$", onda ćemo generirati lanac tako što ćemo prvo primjeniti R:

$$08f9028a \underset{R}{\rightarrow} ntgvkh$$

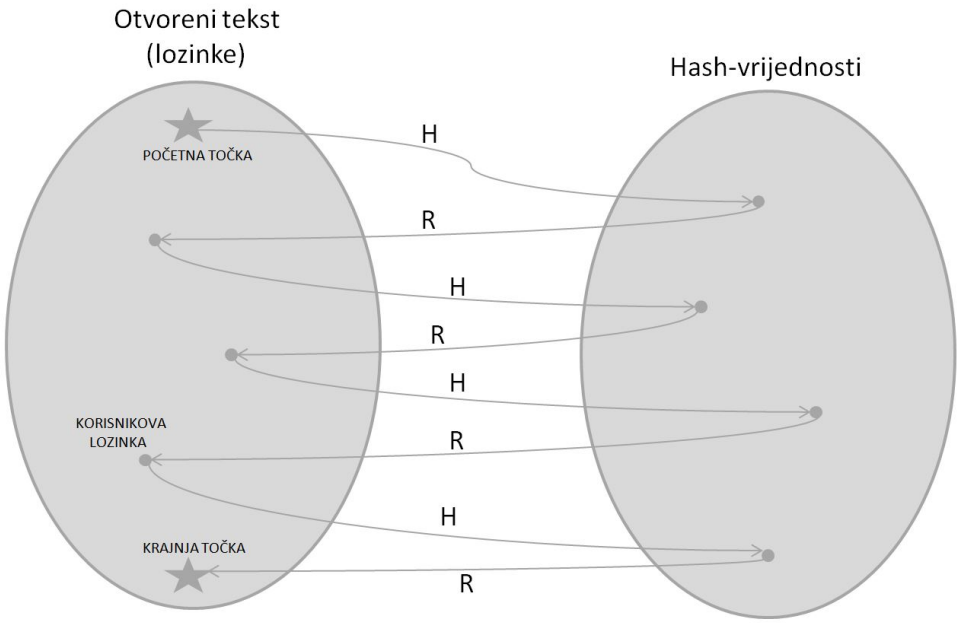
Kako je "*ntgvkh*" jedna od krajnjih točaka u tablici, tada uzimamo odgovarajuću početnu lozinku "*mhmhmh*" te generiramo lanac dok ne dođemo do "$08f9028a$".

$$mhmhmh \underset{H}{\rightarrow} 08ca0280 \underset{R}{\rightarrow} nayxed \underset{H}{\rightarrow} 08f9028a$$

Stoga, tražena lozinka je "*nayxed*" (ili druga lozinka koja ima istu hash-vrijednost).
Međutim, primjetimo da ovaj lanac ne mora uvijek sadržavati hash-vrijednost h. Može se dogoditio da lanac koji počinje sa početnom točkom h se spaja s lancem koji ima drugu početnu točku. Na primjer, moguće je da dobijemo hash-vrijednost "$FB107E70$" te da kada slijedimo njegov lanac dobijemo također "*ntgvkh*":

$$FB107E70 \underset{R}{\rightarrow} bvtdll \underset{H}{\rightarrow} 08e00289 \underset{R}{\rightarrow} ntgvkh$$

No "$FB107E70$" nije lanac koji počinje sa "*mhmhmh*". Takav slučaj zovemo "lažna uzbuna". U tom slučaju zanemarujemo poklapanje (poklapanje hash-vrijednosti i završne točke u tablici) te nastavljamo proširivati lanac kako bi pronašli još jedno poklapanje. Ako lanac proširimo do duljine k bez nađenih "dobrih" podudaranja, onda lozinka nije nikada bila generirana ni u jednom od lanaca.



Slika 3.3: Primjer hashiranja korisnikove lozinke.

Sadržaj tablice ne ovisi o hash-vrijednostima koje ćemo pretvarati. Stvorena je jednom, a zatim se u nepromijenjenom obliku koristi neograničeno puta.
Povećanjem duljine lanca, smanjujemo veličinu tablice te se time također povećava vrijeme potrebno za obavljanje pretraživanja. U jednostavnom slučaju lanaca s vrlo malo elemenata, pretraživanje je vrlo brzo, ali je tablica jako velika. Produljenjem lanaca, pretraživanje se usporava, ali se i veličina tablice smanjuje.
Jednostavni hash-lanci imaju nekoliko nedostataka. Najveći nedostatak je slučaj kada se dva lanca sudare (generiraju istu vrijednost). U tom slučaju se oni spajaju te tablica, unatoč troškovima računanja (vrijeme i memorija), ne pokriva željeni broj lozinki. Taj slučaj je nemoguće učinkovito otkriti jer oba lanca koja su se sudarila nisu spremljena u memoriji. Na primjer, ako treća hash-vrijednost u trećem lancu odgovara drugoj hash-vrijednosti u sedmom lancu, onda će ta dva lanca pokriti gotovo isti niz hash-vrijednosti, ali njihove konačne vrijednosti će biti različite. Hash-funkcija H vrlo vjerojatno neće proizvesti sudar (koliziju) jer se obično otpornost na kolizije smatra važnim svojstvom sigurnosti. Ali funkcija redukcije R, zbog svoje potrebe da pokrije sve moguće otvorene tekstove, ne mora biti otporna na kolizije.
Ostale poteškoće proizlaze iz važnosti odabira ”dobre” funckije redukcije R. Odabirom funkcije R kao identitete dobivamo pristup koji je malo bolji od ”brute-force” pristupa. Tek

kada protivnik ima dobru predodžbu o tome kakva je lozinka, onda može odabrati funkciju redukcije R takvu da su vrijeme i memorija optimalno iskorišteni za takve lozinke, a ne za čitav prostor mogućih lozinki. Ali, funkcija redukcije R ne uzima samo početnu hash-vrijednost nego i ostale hash-vrijednosti iz ostatka lanca te može pretvoriti te ostale hash-vrijednosti natrag u čitav prostor mogućih lozinki (tj. dobivena vrijednost ne mora biti iz "optimalnog" prostora mogućih lozinki). Dakle, može biti teško generirati funkciju redukcije R kako bi se podudarala s očekivanom distribucijom otvorenih tekstova (lozinki).

3.4 Dugine tablice

Dugine tablice učinkovito rješavaju problem kolizija, koji se javlja kod običnih hash-lanaca, zamjenom jedne redukcijske funkcije R sa nizom redukcijskih funkcija $R_1, \ldots, R_k$. Na taj način, kako bi se dva lanca sudarila i spojila, oni moraju imati istu vrijednost u istoj iteraciji. Stoga, završne vrijednosti u svakom lancu će biti identične. Nakon što je kreirana tablica, zadnji korak može biti ponovan prolazak kroz cijelu tablicu te uklanjanje "dupliciranih" lanaca (koji imaju iste završne vrijednosti). Potom se generiraju novi lanci za nadopunjavanje tablice. Ti novi lanci nisu otporni na kolizije (mogu se kratko preklapati), ali se neće spojiti. Time drastično smanjujemo ukupan broj kolizija.
Korištenjem niza redukcijskih funkcija mijenjamo način pretraživanja: budući da hash-vrijednost koja nam je od interesa može biti na bilo kojem mjestu u lancu, pa je stoga potrebno generirati k različitih lanaca. Prvi lanac pretpostavlja da je hash-vrijednost na zadnjoj hash-poziciji u lancu te samo primjenjuje R_k. Sljedeći lanac pretpostavlja da je hash-vrijednost na predzadnjoj hash-poziciji te prvo primjenjuje R_{k-1}, zatim H i na kraju R_k. I tako dalje sve do posljednjeg lanca koji primjenjuje sve redukcijske funkcije naizmjenično sa H. Takav pristup stvara novu "lažnu uzbunu": ako pogrešno pretpostavimo položaj hash-vrijednosti, onda možemo nepotrebno procijeniti lanac.
Premda dugine tablice moraju slijediti više lanaca, to nadomještaju tako što imaju manje tablica. Tablica sa jednostavnim hash-lancima ne može rasti iznad određene veličine bez da brzo ne postaj neučinkovite zbog spajanja lanaca. Kako bi rješili ovaj problem, istodobno se održavaju više tablica, a tada svako pretraživanje mora proći kroz svaku tablicu. Dugine tablice mogu postići slično svojstvo sa tablicama koje su k puta veće, omogućujući im da izvrše za faktor k manje pretraživanja.

3.5 Obrana protiv duginih tablica

Dugine tablice su neučinkovite protiv jednosmjernih hash-vrijednosti koje uključuju duge vrijednosti "soli" (engl. salts). Na primjer, uzmimo u obzir hash-vrijednost lozinke koja je generirana pomoću funkcije *saltedhash* (gdje je "||" operator povezivanja):

$$\text{saltedhash}(\textit{lozinka}) = \text{hash}(\textit{lozinka} \parallel \textit{salt})$$

ili

$$\text{saltedhash}(\textit{lozinka}) = \text{hash}(\text{hash}(\textit{lozinka}) \parallel \textit{salt})$$

Vrijednost "soli" nije tajna te se može generirati na slučajan način i pohranjivati s hash-vrijednosti same lozinke. Velika vrijednost "soli" spriječava napade unaprijed izračunatih algoritama za preračunavanje lozinka, uključujući dugine tablice, tako što osigurava jedinstvenu lozinku za svakog korisnika. To znači da će dva korisnika s istom lozinkom imati različite hash-vrijednosti lozinke (uz pretpostavku da se koriste razčite "soli"). Kako bi protivnik uspio dešifrirati lozinku, on mora generirati tablice za svaku moguću vrijednost "soli".

Za starije Unix lozinke koje koriste 12-bitne vrijednosti "soli", trebamo 4096 tablica što je značajno povećanje troškova za protivnika, ali nije nemoguće sa terabajtnim hard diskovima. SHA2 i byrypt metode, koje se koriste u Linuxu, BSD Unixesu i Solarisu, koriste 128-bitne vrijednosti "soli". Ovako velike vrijednosti "soli" čine nemogućim za probiti bilo kakvim napadom za gotovo sve duljine lozinke. Čak i ako protivnik može generirati milijun tablica u sekundi, trebat će mu bilijun godina za generiranje tablica za sve moguće vrijednosti "soli".

Još jedna tehnika koja pomaže kod sprečavanja napada je istezanje ključa (engl. key stretching). Kada primjenimo istezanje, onda vrijednost "soli", lozinke i neke neposredne hash-vrijednosti nekoliko puta prolaze kroz temeljnu hash-funkciju kako bi se povećalo vrijeme potrebno za izračunavanje svake lozinke. Na primjer, MD5 koristi 1000 iteracija za temeljnu MD5 hash-funkciju, koja kao ulaz uzima vrijednost "soli", lozinku i trenutnu neposrednu hash-vrijednost. Hash-vrijednost lozinke korisnika je povezana od vrijednosti "soli" (koja nije tajna) te krajnje hash-vrijednosti. Dodatno vrijeme prilikom ulogiravanja nije primjetno korisnicima jer moraju pričekati samo djelić sekunde svaki put kada se prijavljuju. S druge strane, istezanje smanjuje učinkovitost "brute-force" napada proporcionalno s brojem iteracija jer smanjuje broj pokušaja koje protivnik može izvršiti u određenom vremenskom okviru. Ovaj princip se primjenjuje u MD5 i u bcrypt. Također, povećava se vrijeme potrebno za generiranje unaprijed izračunatih tablica. No ako maknemo vrijednosti "soli", onda trebamo samo jednom generirati unaprijed izračunate tablice.

Drugačiji pristup, nazvan jačanje ključa (engl. key strengthening), povećava ključ sa slučajnom vrijednosti "soli", ali onda (za razliku od istezanja ključa) sigurno briše vrijednost "soli". Takav postupak prisiljava protivnika i korisnike da izvrše "brute-force"

pretragu kako bi otkrili vrijednost ”soli”.
Dugine tablice, kao i ostali unaprijed izračunati algoritmi za preračunavanje lozinka, ne vrijede protiv lozinki koje sadrže simbole koji su izvan predviđenog raspona ili koje su dulje od onih koje je predvidio protivnik. Međutim, mogu se generirati tablice koje uzimaju u obzir uobičajene načine na koje korisnici pokušavaju odabrati lozinke koje su sigurnije, kao što je dodavanje broja ili specijalnog znaka. Zbog značajnog ulaganja u računalnu obradu, dugine tablice dulje od četrnaest mjesta nisu još uobičajene. Stoga, odabirom lozinke koja je dulja od četrnaest znakova može prisiliti protivnika da pribjegne ”brute-force” napadu.

Bibliografija

[1] J. Katz i Y. Lindell, *Introduction to Modern Cryptography*, Chapman & Hall/CRC, 2008.

[2] A.J. Menezes, P.C van Oorschot i S.A. Vanstone, *Handbook of Applied Cryprography*, CRC Press, 1997.

[3] P. Oechslin, *Making a Faster Cryptanalytic Time-Memory Trade-Of*, dostupno na *https://lasec.epfl.ch/ oechslin/publications/crypto03.pdf* (kolovoz, 2003.)

[4] Wikipedia contributors, *Rainbow table*, dostupno na *https://en.wikipedia.org/w/index.php?title=Rainbow_table&oldid=851448171* (kolovoz, 2018.)

Sažetak

Cilj ovog diplomskog rada je s teorijske strane objasniti sigurnu komunikaciju, integritet poruka, hash-funkcije te dugine tablice.
Prvi dio rada obrađuje dva najvažnija cilja kriptografije, odnosno pružanje privatnosti i integriteta podataka. Sama enkripcija općenito ne pruža integritet podataka, stoga bi se enkripcija trebala koristiti u kombinaciji s drugim tehnikama za postizanje autentifikacije poruke kao što je kod za autentifikaciju poruka, tzv. MAC.
Drugi dio rada se odnosi na hash-funkcije koje uzimaju string proizvoljne duljine te ga "sažimu" u kraći string, tzv. hash-vrijednost. Posebno zanimljive su hash-funkcije otporne na kolizije. Kolizija funkcije H predstavlja par različitih ulaznih podataka x i x' takvih da vrijedi $H(x) = H(x')$. Obično promatramo funkcije koje imaju beskonačnu domenu i konačnu kodomenu pa stoga zahtjevamo da su takve kolizije "teške" za pronaći. "Birthday" napad je općeniti napad koji pronalazi kolizije u svakoj hash-funkciji. Zatim smo predstavili Merkle-Damgardovu transformaciju koja se naširoko koristi u praksi za izradu hash-funkcija otpornih na kolizije. Merkle-Damgardova transformacija omogućuje konverziju iz bilo koje konačnodimenzionalne hash-funkcije u punopravnu hash-funkciju te pritom čuva svojstvo otpornosti na kolizije.
Treći dio rada opisuje dugine tablice koje su razvijene kako bi se lozinka (podatak) izvela direktno iz hash-vrijednosti. Dugine tablice je otkrio P.Oechlin kao primjenu ranijeg algoritma Martina Hellmana. Glavno ograničenje originalne metode Martina Hellmana jest spajanje dva lanca prilikom sudara unutar iste tablice. Dugine tablice učinkovito rješavaju taj problem zamjenom jedne redukcijske funkcije sa nizom redukcijskih funkcija. Dugine tablice su neučinkovite protiv hash-vrijednosti koje uključuju duge vrijednosti "soli". Također, prevencija napada duginim tablica je istezanje ključa te jačanje ključa. S strane korisnika, preporuča se kreiranje lozinki koje su dulje od četrnaest znakova te korištenje simbola koji su izvan predviđenog raspona (npr., @, %, &,. . .).

Summary

The aim of this diploma thesis is to theoretically explain what are secure communication, message integrity, hash-functions and rainbow tables.
The first part of this thesis is concentrated on two most important goals of cryptography, i.e. obtaining private communication and guarantee message integrity. Encryption does not in general provide any integrity, therefore encryption should be used in combination with other techniques for maessage authentication such as Message Authentication Code (MAC).
The second part of this thesis refers to hash-functions that are mapping strings of arbitrary length to strings of some fixed length, called hash-values. Particularly interesting are collision-resistant hash-functions. A collision in a function H is a pair of distinct inputs x and x' such that $H(x) = H(x')$. Usually, we consider functions that have an infinite domain and a finite range, so a requirement is therefore that such collisions should be "hard" to find. "Birthday" attack finds a collision in any hash-function. Then we presented the Merkle-Damgard transform that is used for constructing collision-resistant hash-functions. The Merkle-Damgard transform enables a conversion from any fixed-length hash-function to an arbitrary-length hash-function while maintaining the collision resistance property.
The third part of this thesis describes rainbow tables which have been developed for cracking password directly from hash-values. Rainbow tables were invented by P.Oechlin as an application of an earlier algorithm by Martin Hellman. The main restriction of the original method is the fact that when two chains collide in a single table they merge. Rainbow tables efficiently solve this problem by replacing one reduction function with successive reduction functions. Rainbow tables are ineffective against hash-values that include large "salts". Also, important techniques that help with prevention for those attacks are key stretching and key strengthening. From the side of the user, it is recommended to use passwords that have more than fourteen characters and to use symbols that are out of the usual range (eg., @, %, &,. . .).

Životopis

Rođena sam 08. studenog 1992. godine u Čakovcu, a odrasla sam u Maloj Subotici pokraj Čakovca. Osnovnu školu Mala Subotica upisala sam 1999. godine gdje sam razvila poseban interes za matematiku. Nakon završetka osnovne škole, 2007. godine, upisala sam prirodoslovno-matematičku gimnaziju u Prvoj gimnaziji Varaždin, koju sam završila 2011. godine. Iste godine upisujem preddiplomski studij matematike na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta u Zagrebu. Na trećoj godini studija odabrana sam za demonstratora na kolegiju Teorija brojeva. Preddiplomski studij završila sam 2015. godine te upisujem Diplomski sveučilišni studij Primijenjena matematika, također na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta u Zagrebu. U travnju 2017. godine počinjem s praksom u firmi CloudSense u Zagrebu, gdje ću nastaviti raditi nakon završetka studija.