Kibernetski napadi preko stranskih kanalov

Tjaž Štok, Matevž Pesek
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Večna pot 113, 1000 Ljubljana
ts92284@student.uni-lj.si, matevz.pesek@fri.uni-lj.si

Izvleček

Pri implementaciji kriptografskih operacij v programskih rešitvah pogosto posegamo po optimizaciji programske kode z namenom zmanjšanja dodatnega časa, potrebnega za izračun. Takšne odločitve imajo lahko negativne posledice na uhajanje podatkov izven programske rešitve. Napadalci lahko izkoristijo dodatne informacije, ki nastajajo pri izvedbi programske rešitve, na način, ki jim omogoča dodatne vpoglede v delovanje programa, izračunane vrednosti in dostop do samih podatkov. Vpoglede lahko pridobivajo v obliki časa izvedbe, vzorca dostopa do pomnilnika, porabe energije in drugih, na videz nepovezanih parametrov sistema. Takšne napade imenujemo »napadi preko stranskih kanalov«.

Članek predstavlja več primerov takšnih napadov in razsežnosti uhajanja informacij ter načinov zaščite pred napadi. Opisane so časovna kriptoanaliza, kriptoanaliza na podlagi porabe energije in njena posebna različica – kriptoanaliza na podlagi videoposnetka – ter pomnilniška kriptoanaliza. Na primeru preprostega napada s časovno kriptoanalizo je predstavljen postopek analize in rezultati pred in po implementaciji predlagane zaščite.

Ključne besede: kibernetska varnost, kriptoanaliza, kriptografija, napadi preko stranskih kanalov

Exploiting side-channel attacks in cybersecurity

Abstract

When implementing cryptographic primitives in software, we often resort to optimizing the software code in order to reduce unnecessary computation time. Such decisions can have negative consequences on data leakage outside the implementation. Attackers can exploit the additional information generated by the execution of the software solution in a way that gives them additional insights into the operation of the software, the values calculated and access to the data itself. Insights can be gained in the form of execution time, memory access patterns, power consumption, and other seemingly unrelated system parameters. Such attacks are referred to as »side-channel attacks«.

This paper presents several examples of such attacks and the magnitude of information leakage, as well as how to protect against such attacks. Time-based cryptanalysis, power-based cryptanalysis, including a special variant – video-based cryptanalysis – and memory-based cryptanalysis are described. A simple timing cryptanalysis attack is used as an example to present the analysis process and the results before and after the implementation of the proposed mitigation.

Keywords: cybersecurity, cryptanalysis, cryptography, side-channel attacks

1 UVOD

Kriptografija dandanes postaja vse bolj pomembna stroka, saj digitalni podatki predstavljajo vse od zasebnosti individualne osebe do strogo tajnih državnih skrivnosti. Skozi zadnja desetletja so bili razviti in preizkušeni različni načini zagotavljanja digitalne tajnosti, ki jih v današnjem času uporabljamo že pri vsakdanjih opravilih, kot je opravljanje nakupov preko spleta, dostop do spletne banke in drugih. Kljub temu, da so današnji kriptografski sistemi prestali že veliko napadov in formalnih preverjanj, niso nujno popolnoma odporni na drugačno vrsto napadov: napadi preko stranskih kanalov (angl. *side-channel attacks*). Ti napadi izkoriščajo informacije iz drugih,

nepredvidenih virov (npr. [2]), kot so na primer merjenje časa, merjenje porabe energije, merjenje elektromagnetnega sevanja in drugih zunanjih dejavnikov delovanja sistema.

Namen tega članka je pregled različnih *side-channel* napadov, njihove pomembnosti, načinov izvedbe in možnosti za preprečevanje napadov. Po metodološkem pregledu napadov tega tipa sprva predstavimo časovno kriptoanalizo, ki prikazuje tveganja pri optimizaciji izvajanja ali implementaciji občutljivih kriptografskih operacij. Zatem predstavimo kriptoanalizo na podlagi porabe energije, s katero je poudarjena nezadostnost uporabe konstantno-časovnih algoritmov na modernih centralnih procesnih enotah, saj lahko z analizo porabe dejanske energije in spreminjanja period ure centralne procesne enote glede na porabo spremljamo uhajanje informacij. Nadalje predstavimo kriptoanalizo na podlagi videoposnetka, kjer analiziramo svetilnost LED diode, iz katere je mogoče inducirati porabo energije. Nazadnje na kratko omenimo pomnilniško kriptoanalizo, s katero je zaradi neprevidne implementacije optimizacije predpomnilnika mogoče izluščiti tajne kriptografske ključe iz predpomnilnika centralnih procesnih enot Apple M1.

2 METODOLOGIJA RAZISKOVALNEGA PRISTOPA NAČRTOVANJA IN RAZVOJA

V tej raziskavi smo uporabili raziskovalni pristop načrtovanja in razvoja (angl. *design science methodology* - DSM) za sistematično analizo in ublažitev napadov preko stranskih kanalov na implementacije kriptografskih programov. Proces smo pričeli s temeljito fazo identifikacije problema, pri čemer smo ugotovili, da optimizacija kriptografskega programa za zmanjšanje časa izračunavanja nenamerno uvaja ranljivosti. Te ranljivosti lahko izkoristimo za napade preko stranskih kanalov, kot so časovna kriptoanaliza, kriptoanaliza na osnovi porabe energije, video kriptoanaliza in kriptoanaliza na osnovi pomnilnika. Rezultat identifikacije v tej raziskavi je razviti praktične in učinkovite rešitve, ki ublažijo te vrste napadov, ne da bi pri tem spreminjali delovanja ali namena procesa.

V praktičnem delu članka je naš cilj ustvariti protiukrepe, ki bi učinkovito onemogočili uhajanje informacij skozi stranske kanale. Specifično smo si prizadevali razviti tehnike, ki ohranjajo koristi optimizacije programa, hkrati pa zmanjšujejo tveganje uhajanja informacij. Ukrepe smo načrtovali z namenom uporabe v realnih scenarijih in zagotavljanju robustnih implementacij kriptografskih operacij.

Med fazo načrtovanja in razvoja smo oblikovali različne strategije za ublažitev različnih vrst stranskih napadov. Za časovno kriptoanalizo smo zasnovali algoritme s konstantnim časom in preprečili variacije v času trajanja glede na vhodne podatke, da bi zameglili časovne vzorce. Kriptoanalizo na osnovi porabe energije smo obravnavali z izvajanjem tehnik uravnoteženja porabe energije. Za preprečevanje video kriptoanalize smo poleg nasvetov za preprečevanje kriptoanalize na podlagi porabe energije tudi predlagali ločeno električno vezje za statusne LED diode. Za kriptoanalizo na osnovi pomnilnika smo zagotovili konstantne vzorce dostopa do pomnilnika. Vsaka od teh strategij je bila zasnovana za obravnavo specifičnih značilnosti in ranljivosti, povezanih z ustreznimi vektorji stranskih napadov. Praktičnost in učinkovitost teh rešitev prikazujemo s študijami primerov.

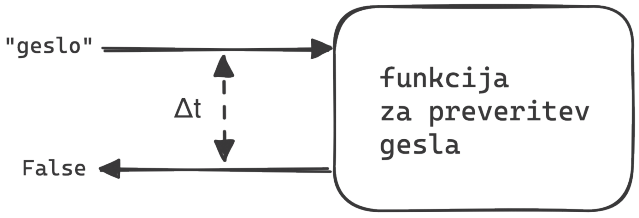
3 PREGLED PODROČJA

Nevarnosti časovne kriptoanalize so sprva demonstrirali Kocher in drugi [4], ki so uspešno pridobili faktorizacijo RSA ključev in Diffie-Hellman eksponente, ta napad pa so kasneje dodelali Dhem in drugi [2] s praktičnim napadom na sistem bralnikov varnostnih kartic. Dovzetnost na časovno kriptoanalizo izhaja iz dolgoletnega truda za boljšo optimizacijo strojne kode, ki jo ustvarijo programski prevajalniki. To je privedlo do razvoja principa izvedbe v konstantnem času, ki zagotavlja, da se kriptografske operacije vedno končajo v približno enakem času ne glede na veljavnost vhodnega podatka [6].

Časovna kriptoanaliza je bila iztočnica za razvoj novih vrst napadov, kot so analiza porabe energije [3][8], ki lahko zaradi narave računalnikov in optimizacije delovanja centralne procesne enote razkrije veliko o stanju kriptografskega sistema. Poleg vpogleda v porabo energije lahko veliko razkrije tudi dostop do (pred)pomnilnika, kar so demonstrirali Vicarte in drugi [7] z napadom *Augury* ter Chen in drugi [1] z napadom *GoFetch*. Razviti so bili tudi bolj eksotični napadi, kot je neinvazivna videoanaliza LED diode bralnika varnostnih kartic [5], ki je omogočila pridobitev tajnega kriptografskega ključa kartic iz bralnika.

3.1 Časovna kriptoanaliza

Časovna kriptoanaliza temelji na statistični analizi časa izvedbe kriptografskih operacij, iz katere lahko ugotovimo njihov notranji mehanizem in v nekaterih primerih celo postopoma sestavimo tajni kriptografski ključ ali geslo. Slika 1 ilustrira abstrakten model tega tipa napada.



Slika 1: **Princip merjenja časa kriptografske funkcije.**

V najpreprostejšem primeru gre za funkcijo, ki preveri, ali je podano geslo pravilno tako, da preveri ujemanje vsakega bajta v pravilnem (shranjenem) geslu. Če naleti na neujemanje bajtov, se funkcija takoj zaključi in vrne rezultat *False*, četudi ni preverila vseh bajtov, saj eno neujemanje že predstavlja napačno geslo. Optimizacija predstavlja tveganje, saj lahko napadalec glede na čas, ki ga funkcija porabi, ugotovi, ali je določen znak pravilen. Če napadalec odkrije ujemanje prvega bajta, lahko opazi, da je funkcija za preverjanje trajala dlje kot prej, saj se je lotila preverjanja naslednjega bajta. Tako lahko napadalec postopoma zgradi geslo zgolj s pomočjo merjenja časa. Algoritem za naivno preverjanje gesla je predstavljen s psevdokodo v Algoritmu 1.

Algoritem 1 Psevdofunkcija za naivno preverbo gesla.

```
1: function check_password(input)
2:     password ← "password"
3:     if |input| ≠ |password| then
4:         return False
5:     for ∀i ∈ {0,1,...,|password|−1} do
6:         if input[i] ≠ password[i] then
7:             return False
8:     return True
```

Preprost, vendar zelo efektiven način ublažitve opisanega napada je izogibanje zgodnjemu vračanju iz funkcije, kadar pride do neujemanja. Če torej pride do neujemanja bajtov v vhodnem geslu, vseeno pustimo da se zanka zaključi in šele nato vrnemo rezultat *False*. Temu konceptu pravimo izvedba v konstantnem času (angl. *constant-time programming*), saj čas izvedbe ni odvisen od vhodnega podatka. Izboljšan algoritem je prikazan v Algoritmu 2 s predpostavko, da je vhodno geslo enake dolžine kot pravilno. V nasprotnem primeru lahko z namenom upoštevanja principa programiranja v konstantnem času preostali prostor zapolnimo s praznimi (*NULL*) znaki.

Algoritem 2 Psevdofunkcija za preverbo gesla v konstantnem času.

```
1: function check_password_constant_time(input)
2:     password ← "password"
3:     result ← |input| = |password|
4:     for ∀i ∈ {0,1,...,|password|−1} do
5:         state ← state ∧ (input[i] = password[i])
6:     return state
```

Alternativno rešitev bi lahko predstavljala tudi zgoščevalna funkcija, ki bi jo uporabili za primerjavo zgoščenih vrednosti pravilnega gesla in vhodnega podatka. Dokler je zgoščevalna funkcija implementirana po principu programiranja v konstantnem času, je časovna analiza primerjave bajtov gesel praktično neuporabna, saj izhodna vrednost kriptografske zgoščevalne funkcije ne poda informacije o vhodni vrednosti.

Morebitnega napadalca lahko upočasni tudi omejevanje števila klicev funkcije v določeni časovni enoti, vendar se je potrebno zavedati, da to ni popolna rešitev in je le dodatek k izvedbi v konstantnem času.

Pri pisanju kriptografskih funkcij je potrebno pozornost usmeriti tudi k programskim prevajalnikom. Slednji lahko namreč nenamerno optimizirajo kodo in s tem prekršijo princip izvajanja v konstantnem času. Prevajalniki so zelo kompleksni programi, ki tudi poskušajo optimizirati kodo zato, da odstranijo potratne ali neuporabljene ukaze, potratno rabo spomina in druge neuporabne operacije. Tak mehanizem optimizacije je marsikdaj (in večinoma) zaželen, vendar lahko v okviru kriptografije povzroči nezanemarljivo škodo. Med prevajanjem programa lahko prevajalnik odstrani namerne poti funkcij ali polnjenje spomina z ničlami (brisanje skrivnih vrednosti), zato so v sistemih in knjižnicah na voljo različni načini preprečevanja prevajalniških optimizacij.

Če želimo kriptografijo uporabiti v praksi, je zelo priporočljiva uporaba preizkušenih in formalno preverjenih knjižnic, saj se s tem lahko izognemo ponavljanju istih napak [6]. Pomembno je tudi temeljito preučiti in preizkusiti vsak del sistema ali programa, saj vsaka nenamerno izdana informacija iz varnega predela naprave (čas poteka, poraba energije, elek-

tromagnetno sevanje in drugi) predstavlja tveganje za napad preko stranskih kanalov.

3.2 Kriptoanaliza na podlagi porabe energije

Alternativni stranski kanal za napad je analiza porabe energije naprave, ki opravlja kriptografske operacije. Leta 1998 jo je demonstriral Kocher [3], ki je s pomočjo diferencialne analize sestavil DES enkripcijske ključe iz bralnikov kartic. Predstavljene so bile tudi rešitve, kot je konstantno-časovna izvedba in vpletanje šuma v energijsko analizo.

Soroden, vendar bolj sofisticiran napad *Hertzbleed* [8] temelji na merjenju porabe energije centralne procesne enote, ki mu sledi izvedba časovne kriptoanalize. Napad je izveden brez fizičnih merilnikov porabe električne energije, saj čas meri s pomočjo vmesnika za merjenje porabe energije v centralni procesni enoti. Pri tem napadu je močno izkoriščen koncept »*dinamično skaliranje frekvence in električne napetosti*« (angl. DVFS; *dynamic voltage and frequency scaling*). DVFS je ključen pri regulaciji temperature in porabe električne energije centralne procesne enote, saj se pri neaktivnosti procesorja frekvenca in napetost znižata, kar zmanjša porabo električne energije. Pri aktivni uporabi procesorja se frekvenca in električna napetost zvišata in tako omogočita hitrejše delovanje centralne procesne enote, znižata pa se pri visoki temperaturi ali električni moči nad določeno mejo z namenom preprečevanja pregrevanja čipov. Ker sta frekvenca in električna napetost odvisna od količine in načinov obdelovanja podatkov, pomeni, da je tudi frekvenca centralne procesne enote odvisna od teh podatkov. Obenem spremenjena frekvenca procesorske ure povzroči drugačen čas izvajanja, kar predstavlja možnost tudi za napad s časovno kriptoanalizo.

Za napad niso potrebni nobeni fizični instrumenti za merjenje porabe energije, kar pomeni, da je napad izvedljiv tudi na daljavo, vendar pod pogojem, da ima napadalec dostop do programskih vmesnikov za merjenje porabe energije in nivo električne napetosti na centralni procesni enoti. Ker je procesorska frekvenca odvisna od aktivnih operacij, so kriptografske operacije s principom izvedbe v konstantnem času dovzetne za napade s časovno kriptoanalizo, saj se glede na podan vhod kriptografski funkciji lahko spremeni procesorska frekvenca (in s tem čas izvedbe).

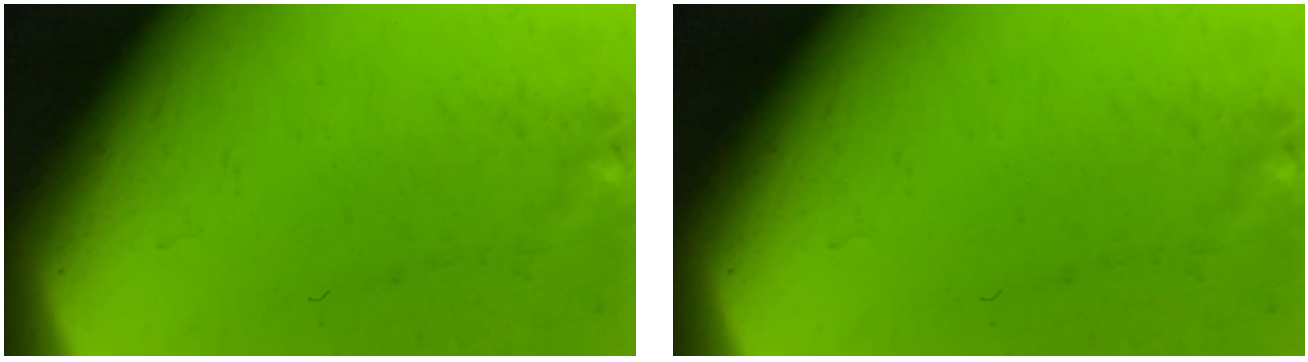
Najpreprostejša ublažitev takega napada je preprečevanje dostopa do vmesnikov za analizo procesorske frekvence in nivoja električne napetosti. Napadalec, ki želi izvesti kriptoanalizo na podlagi porabe energije, bo tako primoran v izvedbo fizičnega merjenja energije, kar je manj natančno in težje izvedljivo. Na področju opisanih napadov so ranljivi vsi moderni Intelovi in AMD-jevi procesorji, vendar obe podjetji priporočata naj avtorji kriptografskih knjižnic sami implementirajo programske popravke glede na izdane smernice.

Ugotovili smo, da so sprejete prakse izvedbe v konstantnem času pomanjkljive in jih je potrebno dodelati, saj lahko procesorji z dinamičnim spreminjanjem frekvence glede na vhodne podatke spremenijo čas izvedbe kriptografskih funkcij, ne glede na to, ali upoštevajo princip programiranja v konstantnem času.

3.3 Kriptoanaliza na podlagi videoposnetka

Tesno povezana s kriptoanalizo na podlagi porabe energije, kriptoanaliza na podlagi videoposnetka napadalcu omogoča izvedbo napada na naprave, ki uporabljajo statusno LED diodo za ponazarjanje delovanja naprave (npr. bralnik brezstičnih varnostnih kartic). Ko taki napravi približamo kartico, naprava prebere tajni kriptografski ključ, s pomočjo katerega opravi kriptografsko operacijo za preverbo veljavnosti kartice. Kadar bralnik sproži kriptografske operacije, se poraba električne energije na krmilniku zviša, zaradi česar se nivo energije na LED diodi zmanjša. Tako majhnih sprememb svetilnosti sicer ne moremo zaznati z golim očesom, lahko pa si pomagamo z video kamerami.

Pri tovrstnem napadu je uporabljena tehnika *rolling shutter*, ki omogoča, da namesto celega okvirja zajamemo več manjših odsekov (vertikalnih vrstic) vsake sličice videa, kar ustvari okoli 60,000 vzorcev na sekundo. Na ta način lahko z veliko višjo frekvenco analiziramo spremembe svetilnosti LED diode, razliko pa je mogoče zaznati s primerjavo sprememb barvnih vrednosti v RGB barvnem prostoru. S pomočjo teh vrednosti lahko induciramo, kakšna je bila sprememba nivoja energije na diodi in s tem zvišanje porabe energije na mikrokrmilniku. Vpogled v te podatke omogoča izvedbo napada na podlagi analize porabe energije (dokazano z napadom *Hertzbleed* [8]) in s tem pridobitev tajnih kriptografskih ključev na varnostnih karticah (npr. 256-bitni ECDSA tajni ključ).



Slika 2: **Razlika v svetilnosti/barvi LED diode med kriptografsko operacijo, ki je nevidna prostemu očesu. Povzeto iz [5] po licenci CC BY-NC.**

V primeru da ima naprava LED diodo vezano ločeno od mikrokrmilnika, je napad mogoč le pod pogojem, da ga preko USB razdelilca priklopimo na drugo napravo, ki ima LED diodo vezano vzporedno z mikrokrmilnikom [5]. Poleg uporabe pametnega telefona je napad izvedljiv tudi s pomočjo IP varnostnih kamer, ki se nahajajo v prostoru, kjer je nameščen bralnik. S predpostavko, da ima napadalec dostop do teh kamer, je napad izvedljiv tudi do 16 metrov razdalje med kamero in bralnikom varnostnih kartic. Postavitev, ki je bila potrebna za napad, je prikazana na Sliki 3.

Slika 3: **Postavitev, namenjena vzorčenju LED diode bralnika varnostnih kartic na oddaljenosti 16 metrov. Povzeto iz [5] po licenci CC BY-NC.**

Ranljivost sistema lahko proizvajalci omejijo tako, da ločijo vezji mikrokrmilnika in LED diode ali pa vzporedno vežejo kondenzator, ki gladi električno napetost, vendar takšne spremembe vodijo v višje cene produktov. Opomniti velja, da pomanjkljivosti ne ležijo v sami LED diodi, ampak v kriptografskih knjižnicah, ki neenotno porabljajo energijo, kar je v opisanem napadu izkoriščeno za analizo in pridobivanje tajnih kriptografskih ključev.

3.4 Pomnilniška kriptoanaliza

Nenehna optimizacija pomnilnikov in predpomnilnikov lahko poleg izboljšav vodi tudi v napake, ki povečujejo ranljivost za napade. Ena izmed najnovejših ponesrečenih optimizacij je bila Applova implementacija tako imenovanega DMP-ja (angl. *Data Memory-Dependent Prefetcher*), ki je iteracijo po tabeli pomnilniških kazalcev pospešil tako, da je v predpomnilnik naložil več podatkov naenkrat. Poleg tega je

vsako vrednost v predpomnilniku, ki je »izgledala« kot kazalec, samodejno naložil [7].

Pri tem procesu problem predstavlja kršitev principa konstantne izvedbe operacij, saj lahko nepredvideno posega po pomnilniku.

Ranljivost je sprva demonstriral napad *Augury* [7], kasneje pa je bila z napadom *GoFetch* [1] izpostavljena še ranljivost tajnih kriptografskih ključev. S tem napadom je kriptoanalizo mogoče izvesti tudi na kriptografskih funkcijah, ki delujejo v konstantnem času.[1]

Za preprečevanje opisanega napada obstajajo trije različni pristopi:

1. **Uporaba »učinkovitih jeder«:** Applovi M1 procesorji so razdeljeni na dve gruči – »učinkovita jedra« (angl. *Efficiency Cores*) in »visokozmogljiva jedra« (angl. *High-Performance Cores*), pri čemer učinkovita jedra nimajo aktiviranega DMP-ja. Izvajanje kriptografskih funkcij bi lahko omejili na ta jedra, vendar bi se zmogljivost pri izvedbi kriptografskih operacij zmanjšala, saj bi se izvajale počasneje. Poleg počasnejšega delovanja bi tvegali tudi morebitne težave zaradi potencialne odločitve proizvajalca za aktivacijo DMP-ja tudi na učinkovitih jedrih.
2. **Slepljenje:** Razvijalci kriptografskih knjižnic bi lahko implementirali princip *»slepljenja«*, ki maskira prave vrednosti kriptografskih skrivnosti, preden so zapisane v pomnilnik. Negativna plat tega pristopa je, da v kriptografske knjižnice prinaša povišano kompleksnost in upočasnitev.
3. **Onemogočanje DMP-ja za nekatere operacije:** Proizvajalec bi lahko omogočil možnost, da razvijalci kriptografskih knjižnic eksplicitno onemogočijo DMP pri opravljanju kriptografskih operacij.

4. PREDSTAVITEV PRIMERA

Med raziskavo smo opravili statistično analizo preproste časovne kriptoanalize, s katero poudarjamo nevarnost takih napadov. Napad smo izpeljali v dveh fazah: v prvi smo poskusili uganiti dolžino gesla, v drugi pa posamezne znake v geslu. Struktura programa je sledeča:

- Funkcija za merjenje časa: S pomočjo Python knjižnice *timeit* merimo čas izvajanja n ponovitev določene funkcije z določenimi argumenti in vrnemo seznam k časov. Iz tega seznama vzamemo najmanjšo vrednost, ki predstavlja lokalno najboljši čas. Funkcija izbrani čas izvajanja določene funkcije izpiše v mikrosekundah.
- Funkcija za ugibanje dolžine gesla: Funkcija izbere diskretni interval, ki predstavlja potencialne dolžine gesla (v našem primeru od 1 do 15 znakov) ter primerja čas izvajanja naključnih gesel (v našem programu smo izbrali nize zapolnjene z n ponovitvami znaka A), pri čemer je niz ob vsaki ponovitvi daljši za en znak (do konca izbranega intervala). Po koncu vzorčenja funkcija izbere geslo, za katerega je funkcija porabila najdlje, s predpostavko, da funkcija za preverjanje gesel takoj zavrne gesla napačne dolžine in preverja le znake vnosov pravilnih dolžin. Funkcija vrne dolžino gesla z najvišjo verjetnostjo pravilnosti.
- Funkcija za ugibanje gesla: Funkcija kot vhod sprejme dolžino gesla, ki smo ga v našem primeru določili s funkcijo za ugibanje dolžine gesla. Pred izbiro znakov je ustvarjen seznam vseh znakov, ki jih lahko sprejme funkcija za preverjanje (v našem primeru male tiskane črke angleške abecede), nato pa v vsaki iteraciji zanke dodamo nov znak na konec gesla, ki ga gradimo. Pri tem merimo čas, ki je porabljen za vsak znak na koncu trenutno zgrajenega gesla (da je preizkusno geslo pravilne dolžine, mu na konec dodamo naključne znake). Kot »pravilen« znak sprejmemo tistega, za katerega je funkcija za preverjanje gesla porabila največ časa, saj sklepamo, da je znak pravilen in se je funkcija za preveritev gesla premaknila na naslednji znak. Ko dosežemo zahtevano dolžino gesla, lahko sklepamo, da smo sestavili geslo, ki je zelo podobno (ali enako) pravilnemu geslu pod pogojem, da smo glede na dolžino gesla izbrali dovolj visoko število vzorcev pri merjenju časa.

[1] Napad je bil uspešno izveden na *OpenSSL*-ovi implementaciji tradicionalne *Diffie-Hellman* izmenjave ključev, Gojeve implementacije RSA dešifriranja *CRYSTALS-Kyber* in *Crystals-Dilithium*.

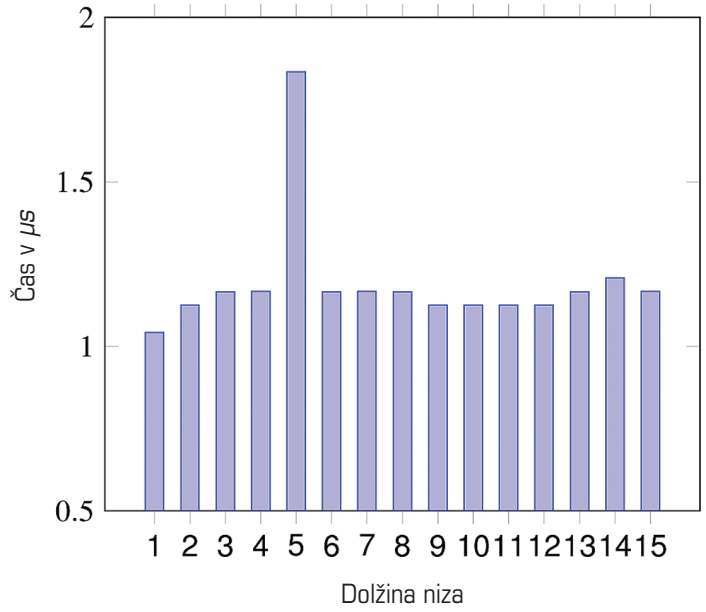
Za tarčo napada smo izbrali Python implementacijo Algoritma 1, ki je prikazana v izvorni kodi 1.

```python
def check_password (input_password: str):

    correct_password = "geslo".encode("utf-8")
    input_password = input_password.encode("utf-8")

    if len(input_password) ≠ len(correct_password):
        return False

    for i in range(len(correct_password)):
        if input_password[i] ≠ correct_password[i]:
            return False

    return True
```

Izvorna koda 1: **Python koda, ki naivno preveri ujemanje gesel.**

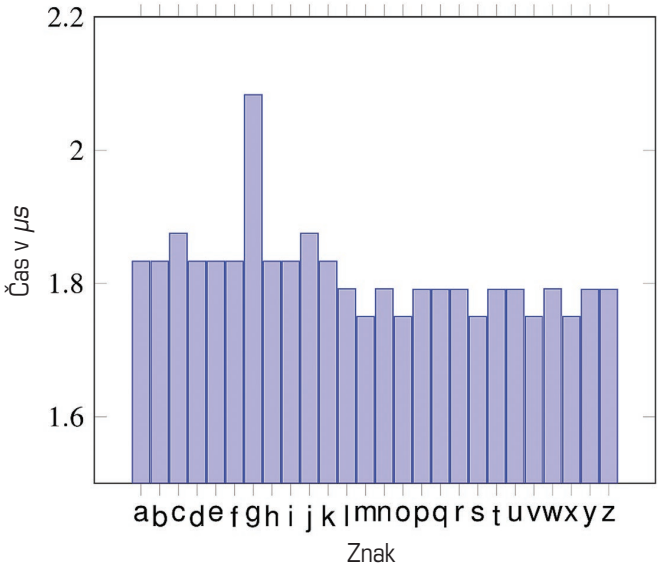
Iz poskusov smo ugotovili, da je za krajša gesla (do 10 znakov) zadovoljiva velikost vzorca $N = 100$, ko gre za poskus ene dolžine ali znaka, kar pomeni grupiranje 10 iteracij funkcije po 10 izvedb poskusov. Iz tega postopka izberemo najmanjšo časovno vrednost, zato da ovržemo odstopanja zaradi nepredvidenih sistemskih upočasnitev. Pri daljših geslih se je natančnost ugibanja nekoliko zmanjšala, vendar je za izboljšanje zadostoval povečan vzorec (do $N = 500$). Obe fazi podrobno analiziramo:

1. Izbrali smo interval dolžin gesla $L = \{1 \leq x \leq 15 | x \in \mathbb{Z}^{0+}\}$ in za vsako diskretno dolžino $l \in L$ preizkusili naključni niz simbolov te dolžine. Po vzorčenju smo s seznama časov izbrali dolžino niza, ki jo je funkcija procesirala najdlje (torej se je dolžina ujemala s pravim geslom in je program preveril ujemanje *vsaj* prvega bajta nizov). Iz Slike 4 je jasno razvidno, da je v našem primeru najverjetnejša dolžina pravega gesla 5.



Slika 4: **Rezultati vzorčenja dolžin naključnih nizov.**

2. Z izbrano dolžino $l \in L$ iz prejšnje faze smo nato pognali proces ugibanja vsakega znaka pravilnega gesla. Funkciji smo z vsako iteracijo posredovali trenutno potrjen niz in naključne znake, s katerimi je bila zapolnjena dolžina. Sprva smo izbrali nabor znakov in nato vsak znak iz množice vzorčili večkrat (npr. $N = 100$; isto kot pri vzorčenju dolžine, 10 zaporednih iteracij funkcije v 10 ločenih izvedbah). Pri tem smo beležili najkrajši čas, ki ga je porabila funkcija za vsak znak (za primerjavo najboljših primerov) in izmed vseh izbrali tistega, za katerega je funkcija porabila največ časa. Izbrani znak smo dodali na konec trenutnega niza in se premaknili na naslednjo iteracijo. To smo ponavljali dokler nismo dosegli dolžine niza, določene v predhodni fazi.



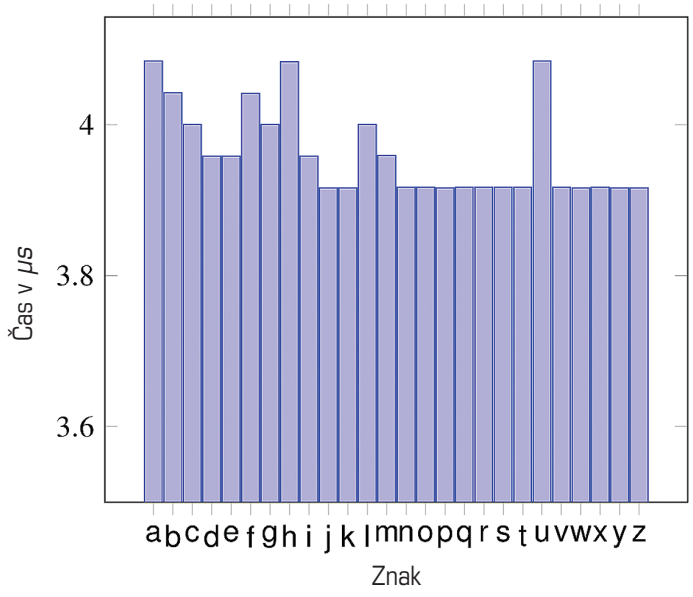
Slika 5: **Rezultati vzorčenja prvega znaka v geslu.**

Iz Slike 5 je razvidno, da je bil v našem primeru najverjetneje prvi znak gesla črka *g*. Ob koncu iteracij napada smo s pomočjo preproste statistične analize in naivno implementirane funkcije našli geslo »*geslo*«. Poskusili smo opraviti tudi časovni napad nad našo kriptografsko funkcijo, ki je izvedena v konstantnem času. Python implementacija funkcije je prikazana v izvorni kodi 2.

Ugotovili smo, da ni mogoče več zanesljivo predvideti, kateri znak je pravilen, saj so bili vsi izmerjeni časi trajanja funkcije zelo podobni in brez očitnega vzorca. Primerjava časovnega poteka funkcije s prvim znakom (s predpostavko, da poznamo dolžino pravilnega gesla) je predstavljena na Sliki 6.

```python
def check_password_constant_time (input_password: str):

    correct_password = "geslo".encode("utf-8")
    input_password = input_password.encode("utf-8")
    result = True

    for i in range(len(correct_password)):
        result &= len(input_password) > i and input_password[i] == correct_password[i]

    return result
```

Izvorna koda 2: **Python koda, ki preveri ujemanje gesel v konstantnem času.**



Slika 6: **Rezultati vzorčenja prvega znaka pri izvedbi v konstantnem času.**

5. DISKUSIJA IN ZAKLJUČEK

V članku smo predstavili različne vrste napadov preko stranskih kanalov, opisali smo njihove značilnosti in načine zaščite pred njimi. Ugotovili smo, da je razlog za njihov obstoj večinoma neprevidna implementacija kriptografskih operacij in agresivna optimizacija prevajalnikov.

Kot prvo smo predstavili časovno kriptoanalizo, utemeljeno na merjenju časa, ki ga porabi kriptografska funkcija. Napadalec lahko s preprostim statističnim modelom predvidi dolžino gesla in znake, ki najverjetneje tvorijo geslo. Kot drugo smo opisali kriptoanalizo na podlagi porabe energije, ki s pomočjo diferencialne analize stanja centralne procesne enote (sprememba električnega toka ali trenutna frekvenca procesorske ure in električna napetost) ter pošiljanja raznih tajnopisov kriptografski funkciji (tako imenovan napad z izbranim tajnopisom) omogoča napadalcu odkritje tajnega kriptografskega ključa. Predstavili smo tudi kriptoanalizo na podlagi videoposnetka, ki s pomočjo pametnega telefona ali IP kamere analizira LED diodo kriptografske naprave (npr. bralnika varnostnih kartic) in inducira porabo energije med izvajanjem kriptografskih operacij, kar omogoči kriptoanalizo na podlagi porabe energije.

Nazadnje smo predstavili tudi pomnilniško kriptoanalizo, s katero je zaradi neprevidne implementacije optimizacije predpomnilnika moč izluščiti tajne kriptografske ključe iz predpomnilnika Applovih centralnih procesnih enot serije M1.

Opravili smo tudi preprost prikaz napada s časovno kriptoanalizo, ki ilustrira pomembnost upoštevanja principov izvedbe v konstantnem času.

S člankom smo želeli izpostaviti dejstvo, da so stranski kanali zelo nepredvidljivi in lahko ogrožajo obstoječe kriptografske sisteme, na katere se vsakodnevno zanaša veliko število uporabnikov. Razumevanje teh napadov je ključnega pomena v kriptografiji, kadar želimo izpopolniti kriptografske funkcije in zmanjševati njihove šibkosti. Hkrati se je pomembno zavedati, da popolnost teoretičnih modelov težko zagotovi popolnoma odporne kriptografske funkcije v praksi, saj napadi preko stranskih kanalov izkoriščajo ranljivosti njihovih implementacij in sistemov, na katerih se le-te izvajajo.

V zadnjih letih se je kriptografija kot veda zelo razširila in dandanes vse več raziskovalcev namenja pozornost odkrivanju novih vrst napadov in zaščit pred njimi. Z večanjem kompleksnosti računalniških sistemov lahko pričakujemo, da bo v prihodnosti prihajalo do vse bolj kompleksnih napadov, ki bodo predstavljali nove izzive pri implementaciji kriptografskih funkcij.

LITERATURA

[1] Boru Chen, Yingchen Wang, Pradyumna Shome, Christopher W. Fletcher, David Kohlbrenner, Riccardo Paccagnella, and Daniel Genkin. Gofetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers. In *USENIX Security*, 2024.

[2] Jean-François Dhem, François Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater, and Jean-Louis Willems. A practical implementation of the timing attack. In Jean-Jacques Quisquater and Bruce Schneier, editors, *Smart Card Research and Applications*, pages 167–182, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[3] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[4] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[5] B. Nassi, E. Iluz, O. Cohen, O. Vayner, D. Nassi, B. Zadov, and Y. Elovici. Video-based cryptanalysis: Extracting cryptographic keys from video footage of a device's power led captured by standard video cameras. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 166–166, Los Alamitos, CA, USA, may 2024. IEEE Computer Society.

[6] A. P. Shivarpatna Venkatesh, A. Bhat Handadi, and M. Mory. Security implications of compiler optimizations on cryptography – a review, 2019.

[7] Jose Rodrigo Sanchez Vicarte, Michael Flanders, Riccardo Paccagnella, Grant Garrett-Grossman, Adam Morrison, Christopher W. Fletcher, and David Kohlbrenner. Augury: Using data memory-dependent prefetchers to leak data at rest. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1491– 1505, 2022.

[8] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. Hertzbleed: Turning power Side-Channel attacks into remote timing attacks on x86. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 679–697, Boston, MA, August 2022. USENIX Association.

▪

Tjaž Štok je študent na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Zanimajo ga področja razvoja programske opreme in kibernetske varnosti. Njegovi raziskovalni interesi zajemajo teorijo kriptografije in razvoja varne programske opreme.

▪

Matevž Pesek je docent in raziskovalec na Fakulteti za računalništvo in informatiko Univerze v Ljubljani, kjer je diplomiral (2012) in doktoriral (2018). Od leta 2009 je član Laboratorija za računalniško grafiko in multimedije. Od leta 2024 izvaja predmet Varnost programov.