UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Računalništvo in matematika – 2. stopnja

Klemen Klanjšček

MENJAVA SPOROČIL MED DVEMA NEZAUPLJIVIMA STRANKAMA

Magistrsko delo

Mentor: prof. dr. Aleksandar Jurišić

Ljubljana, 2019

Zahvala

Zahvaljujem se mentorju prof. dr. Aleksandru Jurišiću za strokovno pomoč, dostopnost in usmerjanje pri pisanju magistrske naloge. Pravtako sem hvaležen tudi asistentu dr. Janošu Vidaliju za vse njegove pripombe in nasvete.

Kazalo

Program dela

Osrednji cilj magistrske naloge naj bo sestava protokola in implementcija varne sheme za izmenjavo sporočil. Osebi želita na pošten način izmenjati vsebino dveh sporočil, ki vstrezata vnaprej postavljenim zahtevam, brez da bi prišlo kadarkoli med izvajanjem protokola do situacije, ko bi bila ena oseba v računsko bistveni prednosti pred drugo. Zgled poštene postopne menjave naj bo izmenjava podpisov pogodbe med dvema nezaupljivima strankama, ki temelji na časovnih prisegah, glej [7]. Za preverjanje specifikacije predlagamo način uporabe Yaovega protokola "popačeno vezje", glej [21] in [42]. V delu predstavite tudi kriptografske osnove, kot so npr. simetrični kriptosistemi, zgoščevalne funkcije, sheme za zaprisego, asimetrični kriptosistemi in varnostna analiza. Naredite tudi osnovno varnostno analizo predlaganih protokolov ter podrobno preučite zmogljivost njihove implementacije v praksi.

Osnovna literatura

[7] D. Boneh in M. Naor, *Timed commitments*, v: Advances in Cryptology — CRYPTO 2000, Springer Berlin Heidelberg, 2000, str. 236–254

[21] M. Jawurek, F. Kerschbaum in C. Orlandi, *Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently*, v: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, ACM, 2013, str. 955–966

[42] A. C.-C. Yao, *How to generate and exchange secrets*, v: 27th Annual Symposium on Foundations of Computer Science, IEEE, 1986, str. 162–167

Podpis mentorja:

Menjava sporočil med dvema nezaupljivima strankama

Povzetek

Namen magistrske naloge je konstruirati in implementirati protokol, ki omogoča dvema nezaupljiva strankama pošteno izmenjavo zasebnih rešitev problemov iz razreda NP brez posredovanja zaupanja vredne tretje osebe. Protokol omogoča preverjanje ustreznosti rešitev problemov iz razreda NP brez razkritja znanja in zagotavlja poštenost menjave, tj. če ena stranka predčasno odstopi od protokola, potem morata obe stranki vložiti primerljivo enako časa za pridobitev zaprošenih rešitev. Verifikacija sporočil temelji na protokolu "popačeno vezje", postopna poštena izmenjava le-teh pa na časovnih zaprisegah. Omejili smo se na izmenjavo rešitev problema iskanja praslik kriptografskih zgoščevalnih funkcij. Pri predpostavki da imamo dovolj dobro mrežno povezavo med sodelujočima, lahko s tako implementacijo izmenjamo krajša sporočila.

Exchange of messages between two mistrusting parties

Abstract

In this thesis we discuss the construction and our implementation of a protocol that allows two independent parties a fair exchange of private solutions of NP problems without a trusted third party. The protocol allows verification of solutions of a NP problem using zero knowledge and ensures fairness of exchange, i.e., if one party quits the protocol early, then the two parties must invest comparable amounts of time to retrieve requested solutions. Verification of solutions is based on the garbled circuit protocol and the fair exchange is formed on timed commitments. Our focus was on exchange of messages with respect to the preimage problem of cryptographic hash functions. Assuming that we have enough bandwidth available we are able to exchange shorter messages.

Math. Subj. Class. (2010): 68P25, 94A60, 11T99

Ključne besede: večstrankarsko računanje, varna evalvacija funkcij, protokol popačeno vezje, dokaz brez razkritja znanja, sheme za zapriseganje, kriptografija javnih ključev, zgoščevalne funkcije

Keywords: multiparty computation, secure function evalutaion, garbled circuit protocol, zero-knowledge proof, commitment schemes, public-key cryptography, hash functions

1 Uvod

V realnem svetu pogosto sodelujemo pri menjavi blaga (ponavadi z osebami, ki jih ne poznamo in jim še ne moremo zaupati) pri čemer želimo, da ob zaključeni menjavi dobimo tisto kar smo izbrali oz. zahtevali (specificirali) in, da imamo možnost preklica menjave, če dobimo nekaj, kar se ne ujema z izbranimi specifikacijami. Enako velja za oba udeleženca v menjavi. Atomarnost takih menjav se ponavadi zagotovi (med drugim) prek zaupanja vredne tretje osebe ZVTO (angl. *trusted third party*), ki deluje kot posrednik. Lahko pa tudi samo z zakoni in drugimi pravnimi elementi. En tak primer (menjave blaga) je prodajanje ali kupovanje prek spletnega podjetja eBay Inc., kjer to podjetje deluje kot zaupanja vredna tretja oseba med dvema udeležencema v menjavi. Take menjave se ponavadi prekličejo z vračanjem fizičnega predmeta pošiljatelju oz. povračila denarja s strani prodajalca ali eBay Inc.

V našem primeru imamo sporočila v digitalni obliki med dvema nezaupljivima anonimnima osebama. Na sporočila lahko gledamo kot na digitalno lastnino. V splošnem take menjave niso prekliclјive, saj se težko zaščitimo proti kopiranju poslanih sporočil. Torej je smiselno, da se sporočila preverijo ali ustrezajo zahtevanim specifikacijam s strani posamezne stranke, še preden se menjava izvede. V primeru da so sporočila v skladu s pričakovanji, želimo, da se menjava tudi v celoti izvede (atomarnost). Obnovitev oz. vračanje (angl. *recovery*) iz "na pol" (npr. samo ena oseba dobi želeno sporočilo) izvedene menjave v splošnem ni mogoča.

V 2. in 3. poglavju bomo formulirali problem, varnostne zahteve in prestavili kriptografske primitive. Reševanje problema, pripadajoče protokole in varnostno analizo le-teh bomo obravnavali v 5., 6. in 7. poglavju. Preostala poglavja (8., 9. in 10.) bomo namenili implementaciji in alternativnim pristopom.

2 Naloga

Naš cilj je konstruirati protokol oz. sistem, ki bo omogočal atomarno menjavo zahtevanih sporočil m_A in m_B med dvema nezaupljivima anonimnima strankama oz. osebama Ana (A) in Bojan (B). Naj ima naslednje lastnosti:

- zagotavlja anonimnost,
- preverjanje zahtev,
- omogočanje izvajanje atomarne menjave samo v primeru da so vse zahteve izpolnjene,
- v protokolu sodelujeta le osebi Ana in Bojan.

2.1 Sporočila in zahteve

Imamo sporočili m_A, m_B, ki sta hranjeni zaporedoma pri osebah A in B. Sporočili sta poljubna niza bitov, tj. $m_A, m_B \in \{0,1\}^*$, ne nujno iste dolžine. Vsaka zahteva mora biti formulirana kot funkcija za preverjanje, tj.

$$\mathsf{Ve}(m; S) : \{0,1\}^* \to \{\texttt{True}, \texttt{False}\},$$

definirana z $\mathsf{Ve}(m; S) := S(m)$, kjer je S specifikacija, ki je računsko izvedljiva v polinomskem času. Take funkcije niso nujno deterministične, saj pri evalvaciji lahko uporabljajo tudi naključne bite z namenom nepristranska preverjanja (pri čemer mislimo na dokazovanje brez razkritja znanja, s čimer se pa mi v resnici ne bomo potrebovali ukvarjati).

Oglejmo si dva primera zahtev:

- Pri poljubnem sporočilu m z ustrezno ℓ-bitno zgostitvijo H_m in neko zgoščevalno funkcijo $H : \{0,1\}^* \to \{0,1\}^\ell$, velja

$$\mathsf{Ve}(m; H(m) \stackrel{?}{=} H_m) = \begin{cases} \texttt{True}, & \text{če } H(m) = H_m, \\ \texttt{False}, & \text{sicer.} \end{cases}$$

- Algoritem za primer izbranega matematičnega problema “preveri” rešitev tega primera v polinomskem času (problemi razreda NP).

3 Kriptografski primitivi

Kriptosistem je peterka $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ z naslednjimi lastnostmi:

- $\mathcal{P}$ je končna množica vseh možnih *čistopisov*,
- $\mathcal{C}$ je končna množica vseh možnih *tajnopisov*,
- $\mathcal{K}$ je končna množica vseh možnih *ključev*,
- $\mathcal{E} = \{E_k \mid k \in \mathcal{K}\}$ je množica *šifrirnih funkcij* $E_k : \mathcal{P} \to \mathcal{C}$ in $\mathcal{D} = \{D_k \mid k \in \mathcal{K}\}$ je množica *odšifrirnih funkcij* $D_k : \mathcal{C} \to \mathcal{P}$ tako, da za vsak $e \in \mathcal{K}$ obstaja $d \in \mathcal{K}$ za katerega velja

$$D_d(E_e(p)) = p, \quad \text{za vsak } p \in \mathcal{P}.$$

Vpeljimo nekaj pojmov, ki so povezani s kriptosistemi.

Kriptosistem je *brezpogojno varen*, če napadalec ne more razbiti sistema tudi z neomejeno računsko močjo. V tem primeru se izkaže, da potrebujemo $|\mathcal{K}| \geq |\mathcal{P}|$ (primer: enkratni ščit, angl. *one-time pad*). Pravimo, da ima kriptosistem s-bitno[1] *statistično varnost*, če je verjetnost uspešnosti napada takega napadalca največ 2^{-s}.

Kriptosistem je *računsko varen*, tudi z najboljšim algoritmom je potrebnih n operacij, da razbijemo sistem. Ponavadi je (vsaj) $n \geq 2^{80}$ ali $n \geq 2^{128}$ (primer: kriptosistem AES). Pravimo, da ima kriptosistem κ-bitno[1] računsko varnost, če je $n > 2^{\kappa}$.

Simetrični kriptosistem je kriptosistem, ki uporablja isti ključ za šifriranje in odšifriranje. Za vsak $k \in \mathcal{K}$ in za vsak $p \in \mathcal{P}$ velja:

$$D_k(E_k(p)) = p.$$

Primeri simetričnih kriptosistemov so Cezarjeva šifra, Vigenèrejeva šifra, AES (Advanced Encryption Standard) in DES (Data Encryption Standard).

Bločne šifre so šifre s vnaprej določeno dolžino čistopisa, tajnopisa in ključa. Bloki predstavljajo čistopis oz. tajnopis. Za bločne šifre želimo naslednje lastnosti:

- odpornost na napade z izbranimi čistopisi oz. tajnopisi, glej [39, §2.2].
- dolžina ključa naj bo majhna (vendar odporno na napad z izčrpnim preiskovanjem),
- učinkovito šifriranje in odšifriranje.

Primera:

- DES: dolžina ključa je 56 bitov, velikost blokov pa 64 bitov.
- AES: dolžina ključa je 128/192/256 bitov, velikost blokov pa 128 bitov.

[1] Definicija bitne varnosti lahko postane hitro dvoumna. Več o tej problematiki, glej Micciancio in Walter [28].

Zgrajena sta na podlagi t.i. substitucijsko-permutacijskih mrež (angl. *substitution-permutation networks*).

Asimetrični kriptosistem je kriptosistem, kjer sta šifrirni in odšifrirni ključ različna. Iz šifriranega ključa ne moremo v doglednem času učinkovito pridobiti odšifrirnega ključa (in obratno). Najbolj znani kriptosistemi (in njihovi matematični problemi, na katerih je zasnovana varnost) so:

- RSA (problem faktorizacije),
- McEliece (problem odkodiranja linearnih kod),
- ElGamal (problem diskretnega logaritma),
- Eliptične krivulje (problem diskretnega logaritma za eliptične krivulje).

Algoritem Rivest-Shamir-Adleman je algoritem, ki spada v družino algoritmov za šifriranje z javnim ključem. Ključe generirano tako, da najprej izberemo različni naključni (močni) praštevil p in q, nato izračunamo javni modul $N = pq$ in izberemo šifrirni eksponent e, za katerega velja $\gcd(e, \varphi(n)) = 1$ (kjer je $\varphi(n) = (p-1)(q-1)$ Eulerjeva funkcija, glej [39, §A.2]) ter izračunamo tak odšifrirani eksponent d, da velja $ed \equiv 1 \pmod{\varphi(n)}$. Na ta način smo pridobili javni ključ (e, N) in zasebni ključ (d, p, q). Šifriranje izvajamo prek funkcije $E_{(e,N)}(x) := x^e \bmod N$, odšifriranje pa prek $D_{(e,p,q)}(y) := y^d \bmod N$, kjer sta $x, y \in \mathbb{Z}_n$.

Problem diskretnega logaritma (angl. *discrete logarithm problem*, kratica DLP) je za dana elementa grupe $\alpha, \beta \in G$, kjer je red elementa α enak $n = |G|$, najti $x \in \{0, \ldots, n-1\}$ tako, da je $\alpha^x = \beta$. Število x se imenuje *diskretni logaritem* elementa β pri osnovi α. Medtem ko je diskretni logaritem (verjetno) težko izračunati (v splošnem), lahko potenco izračunamo učinkovito (z algoritmom kvadriraj in zmnoži, glej [39, §7.6]). Za grupo $(\mathbb{Z}_n, +)$ je diskretni logaritem (deljenje) enostavno izračunljiv, za grupo $\mathbb{Z}_p^*$ pa ne, če so izpolnjeni določeni pogoji (npr. $p-1$ mora imeti vsaj en velik praštevilski faktor z vsaj 224 biti, za druge pogoje glej [39, §6.3.1]).

Kriptografska zgoščevalna funkcija je funkcija

$$H : \{0,1\}^* \to \{0,1\}^\ell \quad (\ell > 0),$$

z naslednjimi lastnostmi:

- funkcija H je deterministična,
- funkcija H je odporna na trčenja (tj. v doglednem času ni moč najti različna x in y, da velja $H(x) = H(y)$),
- za vrednost h je težko najti tak m, da velja $h = H(m)$ (angl. *one-way function*).

Primer so družina kriptografskih zgoščevalnih funkcij SHA (angl. *secure hash algorithm*, kratica SHA).

Idealna zgoščevalna funkcija je funkcija $H : \{0,1\}^* \to \{0,1\}^\ell$, ki je odvisna od zgodovine klicev funkcije. Funkcija za nov element $x \in \{0,1\}^*$, vrne naključno vrednost iz množice $\{0,1\}^\ell$ (privzamemo enakomerno porazdelitev). Če pa je bila

funkcija H v preteklosti že izračunana (klicana) pri argumentu x, potem funkcija H vrne staro vrednost $H(x)$.

Pri $\binom{2}{1}$ *zastrtemu prenosu* (angl. *1-2 oblivious (blindfolded) transfer*, kratica OT) [35], ima pošiljatelj sporočili M_0 in M_1, prejemnik pa bit izbire $c \in \{0, 1\}$. Prejemnik hoče prejeti sporočilo M_c brez, da bi pošiljatelj izvedel njegovo izbiro. Poleg tega hočemo, da prejemnik prejme le izbrano sporočilo. V splošnem imamo lahko $\binom{n}{1}$ zatrt prenos, tj. med n sporočili izbiramo eno.[2]

Shema za zapriseganje je kriptografski primitiv, ki omogoča osebi A, da se zapriseže na neko sporočilo m osebi B z možnostjo kasnejšega razkritja zapriseže vrednosti. Za zaprisego želimo, da izpolnjuje naslednji varnostni zahtevi:

- Zapriseže vrednost c osebi B ne daje nobene informacije o sporočilu m (ali pa zanemarljivo majhno) – *skrivanje*.
- Oseba A ne more razkriti c (ali pa vsaj ne učinkovito) na več različnih načinov – *zaveza*.

Mi se bomo omejili samo na sheme katerih skrivanje in zaveza sta računsko varni. Shemo je mogoče implementirati z uporabo zgoščevalnih funkcij, psevdonaključnih generatorjev števil, pa tudi z drugimi "enosmernimi" funkcijami (enosmerne permutacije, potenciranje, če je DLP računsko prezahteven).

Shema za časovno zapriseganje (angl. *timed commitment scheme*) je shema za zapriseganje, ki omogoča prejemniku zaprisege njeno razkritje, v naprej določenemu končnemu številu operacij. Na ta način rešujemo problem nepravičnosti do prejemnika v primeru navadnih shem za priseganje, tj. čeprav pošiljatelj z zaprisego c zapriseže sporočilo m, prejemnik sporočila m ne more spoznati (zaradi skrivanja), vse dokler pošiljatelj ne razkrije zaprisege (odločitev o razkritju zaprisege je enostranska).

Dokazi brez razkritja znanja (angl. *zero-knowledge proof*). Pri tem kriptografskem protokolu sodelujeta dve osebi, prvi, ki neko lastnost dokazuje (dokazovalec) in drugi, ki trditve prvega preverja (preverjevalec). Oba imata skupen podatek x. Dokazovalec pozna določeno lastnost podatka x in želi preko komunikacije prepričati preverjevalca, da ima podatek x res to lastnost, brez da bi se pri tem preverjevalec naučil karkoli o poznavanju te lastnosti od dokazovalca.

Za dokaze brez razkritja znaja si želimo naslednje lastnosti:

- *uglašenost* (angl. *soundness*) – verjetnost, da preverjevalec sprejme neveljavni dokaz, je zanemarljiva,
- *polnost* (angl. *completeness*) – preverjevalec vedno sprejme veljaven dokaz.

Varno večstrankarsko računanje (angl. *secure multiparty computation*, kratica MPC) je področje kriptografije, kjer pri računanju določene vrednosti sodeluje več udeležencev (naprav). Metoda omogoča skupno izračunavo vrednosti neke funkcije nad zasebnimi vhodi udeležencev, z zagotovilom, da bodo zasebni vhodi ostali "skriti" pred ostalimi udeleženci. Pripadajoče metode spadajo v družino protokolov

[2] V angleščini preberemo $\binom{2}{1}$ kot "2 choose 1", pri čemer beseda *choose* pomeni izbrati. Tako se je prijela okrajšava $\binom{2}{1}$, ki jo v slovenščini pravilno preberemo "izmed dveh izberemo enega".

za *varno računanje funkcij* (angl. *secure function evaluation*, kratica SFE). Primer najbolj znanega protokola za varen izračun funkcije za dve stranki je Yaov protokol *popačeno vezje* (angl. *Yao's garbled circuit protocol*) [42]. Le-ta omogoča skupen izračun poljubne funkcije (ki jo lahko realiziramo z odločitvenim logičnim vezjem) nad zasebnimi vhodi dveh nezaupljivih strank brez posredovanja ZVTO.

3.1 Varnost protokolov

Protokole in njihovo varnost lahko analiziramo tako, da obravnavamo seznam stvari, ki tvorijo varnostne luknje. Problem tega pristopa je okornost in veliko tveganje za napake, zaradi velike količine možnih varnostnih lukenj. Drugi pristop je analiziranje protokola prek primerjanja obnašanja protokola v realnem svetu glede na idealen svet.

- *Idealni svet* (uporaba zaupanja vredne tretje osebe). Funkcionalnost protokola prenesemo na zaupanja vredno osebo. Primer protokola ki zadošča našim potrebam (za pošteno menjavo sporočil) v idealnem svetu, zgleda takole:

  1. Stranki pošljeta ZVTO svoje zahteve in sporočili.
  2. Ko ZVTO dobi obe zahtevi in sporočili, preveri pripadajočo zahtevo nad pripadajočim sporočilom.
  3. Če sta obe zahtevi izpolnjeni, izmenja sporočili in jih pošlje strankama.

  V tem primeru vsa komunikacija med strankama in ZVTO poteka po varnih kanalih.

- *Realni svet* (brez zaupanja vredne tretje osebe). Komunikacija poteka neposredno med udeleženci prek izbranega protokola.

Protokol v realnem svetu je *varen*, če je vsako posledico oz. učinek napadov v realnem svetu možno doseči tudi v idealnem svetu. Več o varnostni analizi protokolov prek primerjave idealnega in realnega sveta, glej Lindell [25].

Glede na obnašanje napadalca ločimo:

- *Na pol pošten napadalec* (angl. *semi-honest attacker*), tj. pošten, vendar radoveden. Napadalec lahko vpliva na sodelujoče v protokolu brez spreminjana poteka protokola. Napadom takega napadalca pravimo *pasivni napadi*.

- *Zlonameren napadalec* (angl. *malicious attacker*). Napadalec lahko vpliva na sodelujoče v protokolu tudi s spreminjanjem poteka protokola. Napadom takega napadalca pravimo *aktivni napadi*.

V primeru dveh udeležencev v protokolu glede na različne vrste napadov in število napadalcev različno obravnavamo varnost protokolov:

- Varnost nas ne zanima oz. jo je nesmiselno obravnavati, v primeru da se oba sodelujoča zlonamerno obnašata.

- Če se noben izmed sodelujočih ne obnaša zlonamerno, potem obravnavamo varnost oz. odpornost protokola le na pasivne napade.

- Če se samo eden izmed sodelujočih obnaša zlonamerno, potem obarvamo varnost oz. odpornost protokola tudi na aktivne napade.

V zadnjem primeru skrbimo za varnost prek prekinitev oz. odstopov. S tem se obvarujemo pred hujšimi posledicami, četudi protokol ni bil uspešno izveden. Če udeleženec zazna goljufijo oz. aktiven napad, odstopi od protokola. V primeru aktivnega napada ko soudeleženec zavrača nadaljnje izvajanje protokola, bo po nekem določenem času čakajoči udeleženec odstopil od protokola. Zaradi takih napadov je "čista" atomarna menjava sporočil nemogoča.

4 Varnostne zahteve

Kot smo omenili, je atomarna menjava nemogoča, zato se bomo zadovoljili s pošteno postopno menjavo. To pomeni, da bomo sporočili postopno izmenjali v fiksnem številu korakov. Pri tem želimo, da sporočilo ostane zakrito do zadnjega koraka. Poleg tega želimo, da nad zakritim sporočilom ni mogoče izvajati vzporedne napade, tj. poizkus razkritja sporočila z grobo silo ne more biti paraleliziran. Če bi postopno izmenjavali bit po bit zasebnih sporočil, se soočimo z naslednjimi problemi:

- Sporočili sta neenake dolžine. Taka menjava ne bo mogoča, če dolžina sporočila m_A ni večkratnik dolžine sporočila m_B ali obratno.
- Vsi biti sporočila niso enako pomembni.
- Z dovoljšno količino bitov delno prejetega sporočila se lahko da rekonstruirati celotno sporočilo s pripadajočim problemom iz razreda NP. Glede na to, da izmenjujemo dve različni sporočili, bo v tem primeru lahko eden od udeležencev v poljubno slabšem položaju. Bodisi zaradi slabše računske zmogljivosti bodisi zaradi različnih specifikacijam pripadajočih problemov iz razreda NP.

Zato bomo pošteno postopno izmenjali le ključe šifriranih zasebnih sporočil in ne sporočil samih.

Preden začnemo s pošteno postopno menjavamo moramo preveriti ali sporočila ustrezajo želenim specifikacijam. V idealnem svetu za to poskrbi ZVTO. V naši situaciji morata udeleženca to storiti sama, pri čemer si ne smeta razkriti zasebnih sporočili. Ta problem bomo rešili z varnim večstrankarskim računanjem.

5 Reševanje naloge

Imamo osebi Ana A in Bojan B, sporočili m_A in m_B in specifikaciji S_A in S_B, ki definirata zahtevi $\mathsf{Ve}(m_A, S_A) = \texttt{True}$ in $\mathsf{Ve}(m_B, S_B) = \texttt{True}$ po sporočilih m_A in m_B. Med samim potekom izmenjave in verifikacije nimamo stikov z ZVTO v katerikoli obliki (centralizirani ali necentralizirani).

Posamezno bomo obravnavali preverjanje in menjavo. Prvi del protokola (verifikacija) bo potekal za vsako osebo (zahtevo) posebej. V tem koraku si osebi izmenjata šifrirana sporočila in šifrirane ključe. Drugi del protokola sestavlja postopno izmenjavo ključev za šifrirana sporočila.

Korektnost šifriranih sporočil in ključev preverita z dokazi brez razkritja znanja (šibkejša različica) za nealgebraične izjave [21]. Te dokaze bomo izvedli s protokolom "popačeno vezje" [16, 11, 9, 3], zastrtim prenosom in zapriseženimi biti. V splošnem (za doseganje pogojev ZKP) morajo biti vsi ti kriptografski primitivi odporni na aktivne napade. Ker je veliko aktivnih napadov (v naši situaciji), bodisi lahko odkriti bodisi v škodo zlonamerne osebe, nam to omogoča uporabo primitivov, ki so odporni le na pasivne napade. Posledično bo protokol enostavnejši z manjšo časovno in prostorsko računsko zahtevnostjo.

Atomarno (postopno) izmenjavo ključev šifriranih sporočil bomo dosegli z izmenjavo podpisov podobe, ki temelji na časovnih zaprisegah [7]. Podpisujemo dve različni pogodbi (šifrirana ključa). Ko udeleženca pridobita podpisa pogodbe, lahko z njima odšifrirata zahtevano sporočilo.

Vsa komunikacija med Ano in Bojanom poteka po varnih kanalih. Opisano funkcionalnost opravlja protokol 1, ki je sestavljen iz enostavnejših delov.

5.1 Ozadje protokolov

V protokolih 1-7 implementiramo pošteno menjavo sporočil m_A in m_B pri specifikacijah S_A in S_B. Glavi je protokol 1, tj. poštena izmenjava sporočil, ki ga sestavljajo protokoli 2, 4 in 6.

Glavna ideja protokola 1 je naslednja:

1. Ana in Bojan vsak zase zašifrirata zasebni sporočili m_A in m_B s simetrično šifro.
2. Z asimetrično šifro zašifrirata ključa K_A in K_B, ki sta bila uporabljena pri šifriranju sporočil m_A in m_B.
3. Med seboj si izmenjata šifrirani sporočili $E_{K_A}(m_A)$ in $E_{K_B}(m_B)$ in šifrirana ključa $E_{\mathrm{pub}_A}(K_A)$ in $E_{\mathrm{pub}_B}(m_B)$.
4. Brez razkritja zasebnih sporočil m_A in m_B in ključev K_A in K_B si dokažeta, da njihovi sporočili ustrezata pripadajočima specifikacijama S_A in S_B in, da sta prejeti šifrirani sporočili $E_{K_A}(m_A)$ in $E_{K_B}(m_B)$ šifrirani s ključema K_A in K_B, ki pripadata prejetim šifriranim ključem $E_{\mathrm{pub}_A}(K_A)$ in $E_{\mathrm{pub}_B}(m_B)$ – 2. del protokola 1.
5. S šifriranimi ključi $E_{\mathrm{pub}_A}(K_A)$ in $E_{\mathrm{pub}_B}(m_B)$ si ključa K_A in K_B postopoma na pošten način izmenjata in s tem pridobita željeni sporočili – 3. del protokola 1.

6 Protokoli

Glavni protokol 1 sestavljajo manjši protokoli, ki v drevesni strukturi zagledajo takole:

```
Protokol 1 - poštena izmenjava sporočil
├── Protokol 2 - generiranije in izmenjava RSA ključev
│   └── Protokol 3 - dokaz brez razkritja znanja za 3 ∤ φ(N)
├── Protokol 4 - popačeno vezje
│   └── Protokol 5 - zastrt prenos
└── Protokol 6 - poštena izmenjava ključev
    └── Protokol 7 - dokaz brez razkritja znanja za Diffie-Hellmanovo
        četverko ⟨g, A, B, C⟩
```

Protokola 1 in 3 sta nova protokola. Protokoli 2, 4, 6 in 7 so modifikacije že obstoječih protokolov. Protokol 5 je enak (z omejitvijo izbire iz $\binom{n}{1}$ na $\binom{2}{1}$) originalu. V nadaljnjih protokolih bomo vpeljali naslednje oznake:

- Bitno dolžino sporočila m bomo označevali z $|m|$.
- S pub_A označujemo javni ključ osebe A.
- Enakomerno naključno vzorčenje elementa α iz množice G označujemo kot $\alpha \in_R G$.
- Z zapisom $K[i : j]$ označujemo podmnožico zaporednih bitov $b_i b_{i+1} \dots b_{j-1}$ δ-bitnega binarnega zapisa števila $K = b_0 b_1 \dots b_{\delta-1}$.
- Z AES-128 označujemo bločno šifro AES s 128-bitnim ključem. Pripadajočo šifro v načinu delovanja števec (angl. *counter mode*, kratica CTR), pa s AES-128-CTR [27].
- Funkcija gcd za naravni števili a in b vrne njun največji skupni delitelj $\gcd(a, b)$.
- Funkcija lcm za naravni števili a in b vrne njun najmanjši skupni večkratnik $\text{lcm}(a, b)$.
- Praštevilska množica $\mathbb{P}$.
- Funkcija lsb za nek bitni niz vrne bit z najmanjšo težo (angl. *least significant bit*).

Protokol 1 Poštena izmenjava sporočil

Vhodi. Udeleženca: Ana in Bojan. Ana ima sporočilo m_A, specifikacijo S_B in Bojan ima sporočilo m_B ter specifikacijo S_A.
Cilj. Ana in Bojan izmenjata sporočili m_A in m_B (Ana prejme sporočilo m_B, Bojan pa m_A), če sta obe specifikaciji izpolnjeni t.i. $\mathsf{Ve}(m_A, S_A) = \texttt{True}$ in $\mathsf{Ve}(m_B, S_B) = \texttt{True}$. Če Bojan oz. Ana predčasno odstopi od protokola, morata oba vložiti primerljivo enako računskega dela za pridobitev zahtevanih sporočil ali pa je pridobitev sporočil računsko neizvedljiva v realnem času.
Protokol:

1. **Priprava.**
   (a) Ana in Bojan se uskladita pri izbiri varnostnih parametrov: $\delta = 1024$, $\kappa = 128$ in $\tau = 80$.
   (b) Ana in Bojan generirata in izmenjata svoja javna RSA ključa $\mathrm{pub}_A := (3, N_A), \mathrm{pub}_B := (3, N_B)$ s protokolom 2.
   (c) Ana izbere inicializacjski vektor $\mathrm{IV}_A \in_R \mathbb{Z}_{2^\tau}$ in ključ $K_A \in_R \mathbb{Z}^*_{N_A}$. Vektor IV_A pošlje Bojanu. Enak postopek izvede Bojan.
   (d) Ana in Bojan si izmenjata velikosti sporočil $|m_A|$ in $|m_B|$.

2. **Preverjanje specifikacij.**
   (a) Ana in Bojan skupaj varno izračunata naslednjo funkcijo

$$F_A : \{0,1\}^{|m_A|+\delta} \to \{0,1\}^{128\lceil |m_A|/128\rceil+\delta+1}, \text{ definirano z}$$
$$(m_A \,||\, K_A) \mapsto \mathsf{AES\text{-}128\text{-}CTR}(m_A; \mathrm{IV}_A, K_A[0:128]) || \, K_A^3 \bmod N_A \,||\, \mathsf{Ve}(m_A; S_A)$$

   prek protokola 4 – popačeno vezje. Le-ta zahteva, da je funkcija F_A predstavljiva z Boolovo logiko oz. odločitvenim logičnim vezjem. V našem primeru je to vezje sestavljeno iz treh večjih delov:
      (i) Logika implementacije funkcije $\mathsf{AES\text{-}128\text{-}CTR}$, ki razdeli sporočilo m_A na bloke $m_b := m[128b, 128(b{+}1)]$ za $b = 0, 1, \ldots, \lceil |m_A|/128\rceil - 1$ dolžine 128 bitov in vsak blok zašifrira na naslednji način:

$$c_b = m_b \oplus \mathsf{AES\text{-}128}(\mathrm{IV} \,||\, 00\ldots b; K_A[0:128]).$$

      Rezultat so bloki c_b. Če dolžina sprožila m_A ni večkratnik števila 128, evalvator sporočilo dopolni z naključim nizom.
      (ii) Logika za izračun K^3 mod N. Uporabimo dve množenji in dve deljenji z ostankom.
      (iii) Logika za evalvacijo $\mathsf{Ve}(m; S)$. Ta preveri specifikacijo S nad sporočilom m in vrne 0, v primeru da specifikacijo ni izpolnjena, drugače vrne 1.

V tem primeru je evalvator vezja Ana, generator vezja pa Bojan. Po končanem izvajanju protokola 4 Bojan pridobi korektne (zanesljive) vrednosti verifikacijske funkcije $\mathsf{Ve}(m_A; S_A)$, šifrirano sporočilo

$$E_{K_A}(m_A) := \mathsf{AES\text{-}128\text{-}CTR}(m_A; \mathrm{IV}_A, K_A[0:128])$$

in šifriran ključ $E_{\mathrm{pub}_A}(K_A) := K_A^3 \bmod N_A$. V primeru da je vrednost verifikacijske funkcije negativna, Bojan odstopi od protokola.

(b) Bojan preveri ali velja $\gcd(E_{\mathrm{pub}_A}(K_A), N_A) = 1$. V nasprotnem primeru Bojan odstopi od protokola.

(c) Ana in Bojan zamenjata vlogi in ponovita preverjanje specifikacij.

3. **Razkrivanje ključev K_A in K_B.**

(a) Ana in Bojan izvedeta protokol 6 za pošteno izmenjavo ključev.

(b) Ko Ana in Bojan pridobita ključa K_A in K_B, lahko šifrirani sporočili $E_{K_A}(m_A)$ in $E_{K_B}(m_B)$ odšifrirata.

Protokol 2 Generiranje in izmenjava RSA ključev

Vhodi. Udeleženca: Ana in Bojan. Javni varnostni parameter $\delta = 1024$.

Cilj. Generiranje in izmenjava veljavnih javnih ključev RSA.

Protokol:

1. **Generiranje RSA ključev**

   (a) Ana izbere praštevili $p_1, p_2 \in_R \mathbb{P}$, po standardu NIST FIPS 186-4 [31]. Pri tem upošteva δ-bitno dolžino javnega modula N_A in šifrirni eksponent 3 z naslednjimi (dodatnimi) lastnostmi za praštevili p_1 in p_2:

   - Praštevili sta varni, tj. $(p_1 - 1)/2, (p_1 - 1)/2 \in \mathbb{P}$ ali pa sta vsaj praštevili za kateri velja $p_1 \equiv p_2 \equiv 3 \pmod 4$.
   - Praštevili sta na intervalu $2^{(\delta-1)/2} \leq p_1, p_2 \leq 2^{\delta/2} - 1$.
   - Njuna absolutna razlika $|p_1 - p_2|$ je večja od $2^{\delta/2-100}$.
   - Ustrezata pogoju $p_1 \not\equiv 1 \pmod 3$ in $p_2 \not\equiv 1 \pmod 3$.
   - Pripadajoči odšifrirni eksponent je dovolj velik, tj.

   $$2^{\delta/2} < 3^{-1} \bmod (\mathrm{lcm}(p_1 - 1, p_2 - 1)).$$

   (b) Ana lahko izračuna javni modul $N = p_1 p_2$ in s tem pridobi javni ključ pub_A. Praštevili p_1 in p_2 shrani za kasnejšo uporabo.

   (c) Ana pošlje svoj javni ključ Bojanu.

   (d) Enak postopek izvede tudi Bojan.

2. **Preverjanje javnih ključev**

   (a) Bojan preveri ali velja $N_A > 2^{\delta-1}$.

   (b) Ana mora Bojanu dokazati brez razkritja vrednosti Eulerjeve funkcije $\varphi(N_A)$ bodisi, da velja $3 \nmid \varphi(N_A)$ (izvedeta protokol 3) bodisi, da je N_A produkt dveh varnih praštevil večjih od 2^ℓ [12].

   (c) Bojan izvede enak postopek za N_B.

Protokol 3 Dokaz brez razkritja znanja za $3 \nmid \varphi(N)$

Vhodi. Udeleženca: Ana in Bojan. Javni podatek je N_A. Ana pozna odšifrirni eksponent $d = 3^{-1} \bmod \varphi(N_A)$.
Cilj. Ana želi Bojanu dokazati, da velja $3 \nmid \varphi(N_A)$ brez razkritja znanja.

Protokol:

1. Bojan izbere $x_B \in_R \mathbb{Z}_{N_A}$. Ta element zapriseže Ani tako, da ji pošlje $c_B = \mathsf{Commit}(x_B, r_B) = H(x_B \,||\, r_B)$, kjer je $r_B \in_R \{0, \ldots, 2^{2\delta}\}$, H pa idealna zgoščevalna funkcija.

2. Ana izbere $x_A \in_R \mathbb{Z}_{N_A}$ in pošlje Bojanu.

3. Bojan izračuna $x := x_A + x_B \bmod N_A$ in preveri ali velja $x \in Z^*_{N_A}$.

4. Bojan izračuna $y := x^3 \bmod N_A$ in ga pošlje Ani.

5. Ana izračuna $x' := y^d \bmod N_A$ in Bojanu pošlje zaprisego $c_A := \mathsf{Commit}(x', r_A)$.

6. Bojan Ani razkrije zaprisego c_B, tj. pošlje ji $x_B \,||\, r_B$.

7. Ana preveri ali velja $(x_A + x_B)^3 \equiv y \pmod{N_A}$ in razkrije zaprisego c_A v primeru, da je konstrukcija vrednosti x bila pravilna.

8. Bojan sprejme dokaz, v primeru da velja $x \equiv x' \pmod{N_A}$, sicer odstopi od protokola.

9. Ana in Bojan ponovita celoten protokol $\lceil \tau / \log_2(3) \rceil$-krat.

Protokol 4 Popačeno vezje

Vhodi. Udeleženca: evalvator (Ana) in generator (Bojan). Evalvator popačenega vezja pri evalvaciji funkcije F prispeva svoj vhod (ključ in sporočilo $K \mid\mid m$). Generator uporabi javne informacije (inicializacijski vektor evalvatorja IV, javni modul evalvatorja N, dolžino sporočila evalvatorja $|m|$ in svojo specifikacijo S) za generiranje popačenega vezja.
Cilj. Varno in zanesljivo izračunati funkcijo F, pri čemer ne razkrijemo zasebnih argumentov.
Potokol:

1. **Priprava.** Generator pripravi Boolovo logično vezje C, ki predstavlja funkcijo F. Vezje sestavljajo dvo-vhodna logična vrata (npr. AND, XOR, OR, ...). Pripadajoče logično vezje C in javne informacije uporabljene pri pripravi vezja deli z evalvatorjem.

2. **Generiranje popačenega vezja.** Generator izvrši naslednje korake.

   (a) Generator za vsako žico (vhodi, izhodi in povezave med logičnimi vrati) w_i vezja C izbere naslednje oznake

   $$w_i^b = (k_i^b \in_R \{0,1\}^\kappa, p_i^b \in_R \{0,1\})$$

   tako, da velja $p_i^b = 1 - p_i^{1-b}$.

   (b) Za vsaka logična vrata G_i v topolškem vrstnem redu vezja C[3] (od vhodov proti izhodom) naredi naslednje:

      (i) Naj bodo G_i logična vrata z dvema vhodoma, ki implementirajo funkcijo $w_c = g(w_a, w_b)$ z vhodnimi oznakami

      $$w_a^0 := (k_a^0, p_a^0),\ w_a^1 := (k_a^1, p_a^1),\ w_b^0 := (k_b^0, p_b^0),\ w_b^1 := (k_b^1, p_b^1)$$

      in izhodnimi oznakami

      $$w_c^0 := (k_c^0, p_c^0) \quad \text{in} \quad w_c^1 := (k_c^1, p_c^1).$$

      (ii) Generira popačeno tabelo za vrata G_i tako, da za vse 4 možne kombinacije vhodnih vrednosti $v_a, v_b \in \{0,1\}$ vrat G_i izračuna

      $$e_{v_a,v_b} := H(k_a^{v_a} \mid\mid k_b^{v_b} \mid\mid i) \oplus w_c^{g_i(v_a,v_b)},$$

      kjer je $H : \{0,1\}^* \to \{0,1\}^\kappa$ idealna zgoščevalna funkcija. vrednosti e_{v_a,v_b} vstavi v tabelo in jih sortira po ključu $(p_a^{v_a}, p_b^{v_a})$.

[3]Na žice lahko gledamo kot usmerjene povezave, ki skupaj z logičnimi vrati oz. vozlišči tvorijo usmerjen aciklični graf. Vsak tak graf lahko topološko uredimo, kar pomeni, da za vsako usmerjeno pot iz vozlišča u v vozlišče v, sledi da je u pred v.

(c) Za vsak izhod iz vezja w_i (izhod vrat G_j) z oznakami $w_i^0 = (k_i^0, p_i^0)$ in $w_i^1 = (k_i^1, p_i^1)$ zgradi dekodirno tabelo za obe možni vrednosti $v \in \{0, 1\}$ izhodne žice w_i tako, da izračuna

$$e_v := \text{lsb}(H(k_i^v \,||\, \text{“izhod”} \,||\, j) \oplus v.$$

vrednosti e_v vstavi v tabelo in jih uredi po ključu p_i^v.

Popačeno vezje $\widetilde{C}$ predstavljajo tabele za posamezna vrata G_i, odkodirne tabele za izhode vezja C in usmerjene povezave med logičnimi vrati.

3. **Evalvacija popačenega vezja.**

(a) Generator pošlje popačeno vezje $\widetilde{C}$ evalvatorju.

(b) Za vsako vhodno žico w_i, ki je namenjena evalvatorjemu zasebnemu vhodu, evalvator in generator izvedeta protokol 5 – zastrt prenos, kjer generator igra vlogo pošiljatelja, evalvator pa vlogo prejemnika. Generatorjeva vhoda v protokol sta zasebni izbiri oznak w_i^0 in w_i^1. Evalvatorjev vhod pa je zasebna bitna vrednost v_i vhodne žice w_i.

Ob izvedbi protokola evalvator pridobi $w_i^{v_i}$.

(c) Evalvator elvalvira prejet $\widetilde{C}$ po nivojih (zvrha navzdol) tako, da začne pri prejetih veljavnih oznakah za svoje vhode. Za vsaka logična vrata G_i s popačeno tabelo $T = (e_{0,0}, e_{0,1}, e_{1,0}, e_{0,1})$ z veljavnimi vhodnimi oznakami $w_a = (k_a, p_a)$ in $w_b = (k_b, p_b)$ izračuna

$$w_c := H(k_a \,||\, k_b \,||\, i) \oplus e_{p_a, p_b}.$$

S tem pridobi veljavno izhodno oznako $w_c = (k_c, p_c)$.

(d) Evalvator odkodira izhod (uporabi odkodirne tabele) in ga primerja s pravim izhodom (v našem primeru vrednost funkcije F_A lahko direktno izračuna, saj so mu znani vsi argumenti). V primeru da se rezultat ne ujema, odstopi od protokola.

(e) Evalvator pošlje neodkodiran izhod, tj. oznake izhodom pripadajočih žic (dokaz o pravilnosti evalvacije) generatorju. Generator odkodira izhod.

Protokol 5 Zastrt prenos

Vhodi. Udeleženca: pošiljatelj in prejemnik. Pošiljatelj ima zasebna vhoda $M_0, M_1 \in \{0,1\}^\ell$, prejemnik pa zasebni bit izbire $c \in \{0,1\}$
Cilj. Prejemnik prejme M_c, brez da bi izvedel M_{1-c} in pošiljatelj ne izve ničesar o vrednosti c.
Protokol:

1. **Priprava.** Izbereta:

   - varnostni parameter κ,
   - aditivno grupo $(G,+) = \langle B \rangle$ reda $p \in \mathbb{P},\ \log(p) > \kappa$,
   - idealno zgoščevalno funkcijo $H : (G \times G) \times G \to \{0,1\}^\kappa$,
   - in simetrično šifro (D, E).

2. **Pridobitev ključev.**

   - Pošiljatelj izbere $y \in_R \mathbb{Z}_p$ in izračuna

   $$S := yB, \quad T := yS.$$

   - Pošiljatelj pošlje vrednost S prejemniku. Ta preveri ali velja $S \in G$ (sicer, odstopi od protokola).
   - Prejemnik izbere $x \in_R \mathbb{Z}_p$ in izračuna

   $$R := cS + xB.$$

   - Prejemnik pošlje R pošiljatelju. Ta preveri, če velja $R \in G$ (sicer, odstopi od protokola).
   - Pošiljatelj izračuna

   $$k_0 := H_{(S,R)}(yR), \quad k_1 := H_{(S,R)}(yR - T),$$

   prejemnik, pa

   $$k_c := H_{(S,R)}(xS).$$

3. **Prenos.**

   - Pošiljatelj zašifrira sporočili M_0 in M_1 zaporedoma s ključema k_0 in k_1

   $$e_0 := E(k_0; M_0), \quad e_1 := E(k_1; M_1),$$

   ter zašifrirani sporočili e_0 in e_1 pošlje prejemniku.
   - Prejemki odšifrira e_c tako, da izračuna

   $$M_c := D(k_c; e_c).$$

Protokol 6 Poštena izmenjava ključev

Vhodi. Udeleženca Ana in Bojan. Javni podatki so pub_A, pub_B, $E_{\text{pub}_A}(K_A)$ in $E_{\text{pub}_B}(m_B)$. Ana ima zaseben ključ K_A in faktorje N_A, Bojan pa zaseben ključ K_B in faktorje N_A.

Cilj. Bojan prejme ključ K_A, Ana pa ključ K_B. Če Bojan oz. Ana predčasno odstopita od protokola, morata oba vložiti primerljivo računskega dela za pridobitev ključev K_A in K_B ali pa je pridobitev le-teh računsko neizvedljiva v realnem času.

Protokol:

1. **Priprava.**

   (a) Ana se z Bojanom dogovori o varnostnih parametrih. Minimalno število $T = 2^\tau = 2^{80}$ modularnih množenj za pridobitev ključa z grobo silo. V primeru da je N_A produkt dveh varnih praštevil večjih od $2^\ell, \ell = 90$, imamo varnostne konstante $P = 3$, $R = 2^{90}$, $Q = 1$, v nasprotnem primeru pa $P = 128$, $R = 2^{10}$ in $Q = 13$.

   (b) Ana izbere $h_A \in_R \mathbb{Z}^*_{N_A}$ Bojan, pa $h_B \in_R \mathbb{Z}^*_{N_B}$. Ti števili si hitro izmenjata (časovna omejitev npr. $\Delta t = 1\text{s}$).

   (c) Ana izračuna $g_A = h_A^{\prod_{i=0}^{r} q_i^{\delta/2}} \bmod N_A$, kjer so $q_0, q_1, ...q_r$ vsa praštevila manjša od P.

   (d) Ana pošlje Bojanu h_A in g_A. Bojan preveri ali je produkt g_A pravilno konstruiran.

   (e) Ana izračuna vektorja $W_A := \langle u_0, u_1, \ldots, u_\tau \rangle$ in $Z_A := \langle v_0, v_1, \ldots, v_\tau \rangle$ kot:

$$Z_A = \left\langle g_A^2, g_A^4, g_A^{16}, g_A^{256}, \ldots, g_A^{2^{2^i}}, \ldots, g_A^{2^{2^\tau}} \right\rangle \bmod N_A,$$

$$W_A = \left\langle g_A^{3\cdot 2}, g_A^{3\cdot 4}, g_A^{3\cdot 16}, g_A^{3\cdot 256}, \ldots, g_A^{3\cdot 2^{2^i}}, \ldots, g_A^{3\cdot 2^{2^\tau}} \right\rangle \bmod N_A.$$

   (f) Ana pošlje vektor W_A Bojanu. Ana dokaže s protokolom 7 – dokaz brez razkritja znanja za Diffie-Hellmanovo četverko $\langle g, A, B, C \rangle$ [13], da je trojica (g_A, u_{i-1}, u_i) oblike $(g_A, g_A^{3x}, g_A^{3x^2})$ za vsak $(1 \leq i \leq \tau)$.

   (g) Bojan opravi "enake" korake priprave kot Ana, (tj. vloga Ane in Bojan je zamenjana) in s tem Ana pridobi vektor W_B.

(h) Ana Bojanu pošlje

$$V_A := K_A \prod_{i=0}^{\tau} v_i^{(\text{Ana})} \bmod N_A,$$

Bojan pa preveri ali velja

$$V_A^3 \equiv E_{\text{pub}_A}(K_A) \prod_{i=0}^{\tau} u_i^{(\text{Ana})} \pmod{N_A}.$$

Vlogi Ane in Bojana se zamenjata in tekoči korak se ponovi.

2. **Postopna izmenjava.**

Ana in Bojan izmenično razkrivata komponente vektorjev Z_A in Z_B.

(a) Ana razkrije Bojanu $v_\tau^{(\text{Ana})}$. Bojan preveri $(v_\tau^{(\text{Ana})})^3 \equiv u_\tau^{(\text{Ana})} \pmod{N_A}$ ter odgovori z $v_\tau^{(\text{Bojan})}$ in $v_{\tau-1}^{(\text{Bojan})}$. Ana preveri prejeto ter pošlje $v_{\tau-1}^{(\text{Ana})}$ in $v_{\tau-2}^{(\text{Ana})}$.

(b) Na ta način postopno izmenjujeta vrednosti v_i vse dokler ne izmenjata vseh (v $\tau + 1$ interakcijah).

(c) Ana ima sedaj cel vektor Z_B in lahko izračuna

$$K_B := V_B \Big/ \prod_{i=0}^{\tau} v_i^{(\text{Bojan})} \bmod N_B.$$

Enako velja za Bojana.

3. **Pridobitev ključa K z grobo silo.**

Če Ana med postopno izmenjavo predčasno odstopi od protokola (t. i. Ana je razkrila le $j < \tau$ elementov vektorja V_A), lahko Bojan pridobi ključ K_A tako, da izračuna

$$v_i^{(\text{Ana})} := g_A^{2^{2^i}} \bmod N_A \quad (1 \le i \le \tau - j).$$

Bojan potrebuje izračunati približno $2^{\tau-j}$ modularnih množenj. Ana pa ima lahko le en element več kot Bojan. To pomeni, da ima najmanj dvakrat manj računskega dela, kot Bojan za pridobitev ključa K_B.

Protokol 7 Dokaz brez razkritja znanja za Diffie-Hellmanovo četverko $\langle g, A, B, C\rangle$

Vhodi. Udeleženca: Ana in Bojan. Javni podatki so N_A, g_A, W_A ter varnostna parametra R in Q. Ana pozna red elementa g_A v $\mathbb{Z}^*_{N_A}$, tj. $\mathrm{ord}(g_A) = q$ ali pa vsaj vrednost $\varphi(N_A)$.
Cilj. Ana želi Bojanu dokazati, da je trojica (g_A, u_{i-1}, u_i) oblike $(g_A, g_A^{3x}, g_A^{3x^2})$ za vsak $1 \leq i \leq \tau$) brez razkritja znanja.
Protokol:

1. Bojan izbere vrednosti $c_1, \ldots, c_\tau \in_R \{0, \ldots, R\}$. Te elemente zapriseže Ani, tako da ji pošlje $c_B = \mathsf{Commit}(c_1, \ldots, c_\tau, r) = H(c_1 \,||\, \ldots \,||\, c_\tau \,||\, r)$, kjer je $r \in_R \{0, \ldots, 2^{2\tau\lceil \log_2 R\rceil}\}$ in H idealna zgoščevalna funkcija.

2. Ana izbere $\alpha_1, \ldots, \alpha_\tau \in_R \mathbb{Z}_q$ in izračuna

$$z_i := g_A^{\alpha_i} \bmod N_A \quad \text{ter} \quad w_i := u_{i-1}^{\alpha_i} \bmod N_A \qquad (1 \leq i \leq \tau).$$

   Potem pošlje pare $\langle z_i, w_i\rangle_{i=1}^{\tau}$ Bojanu.

3. Bojan odpre zaprisego in Ana pridobi vrednosti $c_1, \ldots, c_\tau$.

4. Ana izračuna $y_i := c_i \cdot 3 \cdot 2^{2^{i-1}} + \alpha_i \bmod q$ $(1 \leq i \leq \tau)$ in jih pošlje Bojanu.

5. Bojan preveri ali velja

$$g_A^{y_i} \cdot u_{i-1}^{-c_i} \equiv z_i \pmod{N_A}, \quad u_{i-1}^{y_i} \cdot u_i^{-3c_i} \equiv w_i \pmod{N_A} \quad (1 \leq i \leq \tau).$$

6. Ana in Bojan protokol ponovita Q-krat.

7 Varnostna analiza

Zaradi zamudnega dokazovanja pravilnosti protokolov z uporabo t.i. simulatorjev [25] bomo varnost pretežno obarvali tako, da bomo preučili seznam stvari, ki tvorijo varnostne luknje. Varnost "novih" protokolov bomo v tem smislu neformalno dokazali. Za modificirane protokole bomo podali referenco na ustrezen dokaz in komentirali, zakaj je varnost po našem mnenju še vedno zagotovljena.

7.1 Protokol 1 – poštena izmenjava sporočil

V pripravi protokola 1 smo za bitno dolžino javnih modulov N_A in N_B v kriptosistemu RSA izbrali vrednost $\delta = 1024$. Ta izbira zadošča minimalni računski varnosti po standardu FIPS 168-4. Lahko bi izbrali varnejšo različico, tj. $\delta \in \{2048, 3072\}$, ampak smo zaradi zmogljivostnih omejitev pri procesu razvoja realne realizacije protokola ostali pri prvotni izbiri.

Za varnostni parameter κ, ki predstavlja tako bitno dolžino ključa pri simetričnem kriptosistemu AES, kot bitno dolžino oznak pri protokolu "popačeno vezje", bi lahko izbrali varnejšo različico, tj. $\kappa = 256$, ampak smo se zaradi podobnih razlogov, kot pri izbiri parametra δ, odločili ostati pri izbiri, ki zadošča minimalni računski varnosti po standardu FIPS 197, tj. $\kappa = 128$.

Varnostni parameter $\tau = 80$ določa bitno dolžino inicializacijskega vektorja IV, število zaporednih množenj (2^{80}) za pridobitev ključa, takoj po izmenjavi zakritih ključev V_A in V_B pri protokolu 6 – "poštena izmenjava ključev" in zgornjo mejo za verjetnost goljufije 2^{-80} pri dokazih brez razkritja znanja (protokola 3 in 7).

Pri izmenjavi javnih parametrov so možni naslednji aktivni napadi:

- Če Ana ali Bojan izbere ključ $K \notin \mathbb{Z}_N^*$, bo taka goljufija zaznana pri preverjanju specifikacij, ko se preveri $\gcd(E_{\text{pub}}(K), N) = 1$), saj velja:

$$K \notin \mathbb{Z}_N^* \Rightarrow K^3 \bmod N \notin \mathbb{Z}_N^*.$$

- Če Ana ali Bojan pošlje lažni inicializacijski vektor IV, bo to storil v svojo škodo, saj "ne bo mogel" preveriti izhoda popačenega vezja.

- Če Ana ali Bojan pošlje lažno bitno dolžino sporočila $|m|$, bo goljufija zaznana pri pri testiranju specifikacij.

Varnost računanja funkcije F se obravnavna v podpoglavju 7.4. Nadaljnja analiza predpostavlja, da je bila funkcija F varno izračunana (evalvirana v idealnem svetu).

Pasivni napadi med preverjanjem (situacija je enaka, če sta vlogi Ane in Bojana zamenjani vendar tega ne bomo posebej navajali):

- Ali lahko Bojan pridobi ključ $K_A[0:128]$ (del ključa K_A za simetrično šifriranje sporočila m_A) iz vrednosti $K_A^3 \bmod N_A$ z uporabo že znanih napadov na RSA [8] (pri varni izbiri javnega modula N_A)? Bojan ne more uporabiti celoštevilskega korenjenja za pridobitev ključa $K_A[0:128]$, saj je le del "naključnega" 1024 bitnega niza (angl. *random padding*). Za splošno uporabo

kriptosistema RSA ne dovoljujemo šifrirnih eksponentov manjših od $2^{16}+1$. S tem omejimo napade, ki temeljijo na Coppersmithovemu izreku [8, §4]. Ker v našem primeru naključno izbrani ključ K_A šifriramo le enkrat (ključ K_A je samo za enkratno uporabo), so omenjeni napadi povezani z majhnih šifrirnim eksponentom nepraktični.

- Ali lahko Bojan pridobi ključ $K_A[0:128]$, če pozna nekaj blokov m_b sporočila m_A? Ne, ker je kriptosistem AES odporen na napade z izbranimi čistopisi oz. tajnopisi (v tem primeru imamo premalo takih parov – čistopis / tajnopis).

Aktivni napadi med preverjanjem:

- Če Ana poda sporočilo, ki ne ustreza specifikacijam, bo goljufija zaznana, ko Bojan pridobi vrednost verifikacijske funkcije $\mathsf{Ve}(m_A; S_A)$.
- Aktivni napadi s strani Bojana so del varnosti računanja funkcije F_A.

Preostane nam še razkrivanje ključev K_A in K_B. Varnost poštene izmenjave ključev se obravnava v podpoglavju 7.6. Če se je poštena izmenjava uspešno zaključila, lahko odšifrirata zahtevani sporočili, tj. $E_{K_A}(m_A)$ in $E_{K_B}(m_B)$. V nasprotujem primeru morata za pridobitev ključev K_A in K_B oba vložiti primerljivo računskega dela ali pa je pridobitev le-teh računsko neizvedljiva v realnem času.

7.2 Protokol 2 – generiranije in izmenjava RSA ključev

Generiranije ključev izvajamo po standardu FIPS 186-4. Pri tem upoštevamo δ-bitno dolžino javnega modula N in šifrirni eksponent 3. Poleg pogoja, da sta p_1 in p_2 različni praštevili enake bitne dolžine, pogoj $p_1 \equiv p_2 \equiv 3 \pmod 4$ zagotovi, da je javni modul $N = p_1 p_2$ t.i. Blumovo število. Iz tega sledi, da je problem iskanja inverza *trapdoor* funkcije $f : x \mapsto x^2 \bmod N$ enako težek kot problem ločevanja med kvadratni in nekvadratni ostanki v Z_N, Blum [6]. Varnost protokola 6 temelji na varnosti generatorja Blum–Blum–Shub, tj. generator $x_{i+1} := x_i^2 \bmod N$, kjer je N Blumovo število in x_i kvadratni ostanek v $\mathbb{Z}_N^*$. Za varnost protokola 6 je pomembno, da ima zaporedje Blumovega generatorja

$$\left\{ g_A^{2^i} \bmod N_A \right\}_{i=0}^{\infty}$$

dolgo periodo pri naključno izbranemu začetnemu ostanku g_A. Pri tako izbranih parametrih je dolžina takega zaporedja v povprečju precej dolga. To je posledica izreka, ki pravi, da je Blum–Blum–Shub psevdo-naključnostni generator "nepredvidljiv", glej Blumm et al. [5, §7].

Carmichaelova funkcija $\lambda : \mathbb{N} \to \mathbb{N}$ vsakemu naravnemu številu N priredi najmanjše naravno število $\lambda(N)$, za katerega velja

$$\forall a \in \mathbb{Z}_N^*,\ a^{\lambda(N)} \equiv 1 \pmod N.$$

V splošnem je pomembno, da je vrednost $\lambda(\lambda(N))$ veliko število, saj je večkratnik (in zgornja meja) dolžne periode psevdonaključnega generatorja Blum–Blum–Shub v $\mathbb{Z}_N^*$. Meja je dosežena, če je red elementa 2 v $\mathbb{Z}^*_{\lambda(N)/2}$ enak vrednosti $\lambda(\lambda(N))$ in smo za seme generatorja vzeli kvadratni ostanek, katerega red je $\lambda(N)/2$ v $\mathbb{Z}_N^*$, glej [5, §8]. Tako periodo dobimo z naslednjo izbiro parametrov:

- Za modul $N := p_1 p_2$ izberemo produkt dveh različnih *posebnih* praštevil. Posebno praštevilo p je praštevilo oblike $p := 2q + 1 = 2(2r + 1) + 1$, kjer sta q in r tudi praštevili. Uporabljali bomo naslednje oznake za praštevila modula N iz te konstrukcije:

  $$p_1 := 2q_1 + 1,\ q_1 := 2r_1 + 1,\ p_2 := 2q_2 + 1 \ \text{ in } q_2 := 2r_2 + 1.$$

  Posledično za vrednosti $\lambda(N)$ in $\lambda(\lambda(N))$ velja

  $$\lambda(N) = \operatorname{lcm}(p_1 - 1, p_2 - 1) = 2q_1 q_2 \quad \text{ in } \quad \lambda(\lambda(N)) = \operatorname{lcm}(q_1 - 1, q_2 - 1) = 2r_1 r_2$$

  .

- Praštevili r_1 in r_2 je potrebo izbrati tako, da je modu N δ-bitne dolžine in da, velja

  $$q_1 \not\equiv 1 \text{ ali } 7 \pmod 8 \quad \text{ ali } \quad q_2 \not\equiv 1 \text{ ali } 7 \pmod 8. \tag{7.1}$$

  Lema 7.1. *Zgornja izbira praštevil q_1 in q_2 nam zagotovi, da je red elementa 2 v $\mathbb{Z}^*_{\lambda(N)/2}$ enak $\lambda(\lambda(N))$.*

  Dokaz. Red elementa 2 v $\mathbb{Z}^*_{\lambda(N)/2}$ deli $2r_1r_2$, ker velja $\lambda(\lambda(N)/2) = \lambda(\lambda(N))$. Predpostavimo, da je red elementa 2 v $\mathbb{Z}^*_{\lambda(N)/2}$ ni $2r_1r_2$. Potemtakem sledi, da je njegov red bodisi enak vrednosti r_1r_2 bodisi deli vrednost $2r_1$ ali $2r_2$. Ločimo tri primere:

  - Red elementa 2 v $\mathbb{Z}^*_{\lambda(N)/2}$ deli vrednost $2r_1$. Potem je $2^{2r_1} \equiv 1 \pmod{q_1q_2}$ iz katerega sledi $2^{2r_1} \equiv 1 \pmod{q_2}$. Po Fermatovem izreku dobimo, da je $2^{q_2-1} \equiv 2^{2r_2} \equiv 1 \pmod{q_2}$. Potemtakem je red elementa 2 v $\mathbb{Z}^*_{q_2}$ enak $\gcd(2r_1, 2r_2) = 2$. To je protislovje, saj je q_2 večje od 3.
  - Red elementa 2 v $\mathbb{Z}^*_{\lambda(N)/2}$ deli vrednost $2r_2$. Tudi tu podobno sklepamo kot v prejšnji točki, le da vloge praštevil $\{r_1, q_1\}$ in $\{r_2, q_2\}$ zamenjamo.
  - Red elementa 2 v $\mathbb{Z}^*_{\lambda(N)/2}$ je enak vrednosti r_1r_2. Potem je

    $$2^{r_1r_2} \equiv 1 \pmod{q_1q_2}.$$

    Od tod sledi

    $$2^{r_1r_2} \equiv 1 \pmod{q_2}.$$

    Iz tega lahko sklepamo, da je $2^{r_2} \not\equiv -1 \pmod{q_2}$, ker velja

    $$(-1)^{r_1} \not\equiv (2^{r_2})^{r_1} \pmod{q_2},$$

    saj je praštevilo r_1 je liho. Zato je

    $$2^{(q_2-1)/2} \not\equiv -1 \pmod{q_2}$$

    kar pomeni, da je število 2 je kvadratni ostanek v $\mathbb{Z}^*_{q_2}$. Na enak način lahko pokažemo, da je 2 kvadrirani ostanek tudi v $\mathbb{Z}^*_{q_2}$. Ker pa je zaradi pogoja 7.1 vsaj za eno izmed grup $\mathbb{Z}^*_{q_1}$ in $\mathbb{Z}^*_{q_2}$ element 2 ni kvadratni ostanek, smo prišli do protislovja.

V vseh treh primerih smo prišli do protislovja, zato je začetna predpostavka napačna in velja njen obrat, tj. vrednost $\lambda(\lambda(N)) = 2r_1r_2$ je red elementa 2 v $\mathbb{Z}^*_{\lambda(N)/2}$. □

- Če izberemo za seme generatorja kvadratni ostanek x_0 v $\mathbb{Z}^*_N$, za katerega velja

$$x_0^{q_1} \not\equiv 1 \pmod{N} \quad \text{in} \quad x_0^{q_2} \not\equiv 1 \pmod{N},$$

je njegov red enak vrednosti $q_1q_2 = \lambda(N)/2$. Ker je red semena x_0 v $\mathbb{Z}^*_N$ enak vrednosti q_1q_2 in je red elementa 2 v $\mathbb{Z}^*_{q_1q_2}$ enak vrednosti $2r_1r_2$, ima zaporedje pripadajočega Blum–Blum–Shubovega generatorja periodo dolžine $\lambda(\lambda(N)) = 2r_1r_2$.

Pri implementaciji protokola 2 konstrukcije modula N iz posebnih praštevil nismo upoštevali, saj je pri izbranih varnostnih prametrih za protokol 6 dovolj, če je perioda vsaj $2^{2\tau} = 2^{160}$. Poleg tega lahko iskanje posebnih praštevil traja več ur. V primeru računanja 1024-bitnega modula N sestavljenega in posebnih praštevil v okoliščinah opisanih v poglavju 10 potrebujemo v povprečju 1.6 ure.

V povezavi z generiranjem RSA ključev so "možni" naslednji aktivni in pasivni napadi:

- Če Ana ali Bojan izbere javni modul N, za katerega velja $3 \mid \varphi(N)$, bo ta goljufija zaznana pri preverjanju javnega ključa. Če ne bi preverjali ustreznosti konstrukcije javnega ključa, bi lahko napadalec zakril oz. razkril napačen ključ pri protokolu 6.

- Če bi bila vrednost $\lambda(\lambda(N_A))$ majhna (v smislu računske varnosti), bi lahko napadalec poiskal cikel dolžine $x \leq \lambda(\lambda(N_A))$ v zaporedju $\{g_A^{2^i} \bmod N\}_{i=0}^{\infty}$. Dolžino cikla bi uporabil za učinkovit izračun vrednosti $g_A^{2^{2^\tau}} \bmod N_A$ tako, da bi najprej zmanjšal eksponent, tj. $e = 2^\tau \bmod x$ in potem izračunal $g_A^{2^e} \bmod N_A$, glej protokol 6.

- Z nepravilno konstrukcijo javnega ključa, se njegov lastnik kvečjemu izpostavi napadom povezanih s faktorizacijo.

7.3 Protokol 3 – dokaz brez razkritja znanja za $3 \nmid \varphi(N)$

Protokol temelji na naslednji trditvi 7.2.

Trditev 7.2. *Če $3 \mid \varphi(N)$, potem za vsak kubični ostanek $x \in \mathbb{Z}^*_N$ obstajajo vsaj trije različni kubični koreni.*

Dokaz. Če $3 \mid \varphi(N)$, potem obstaja element $a \in \mathbb{Z}^*_N$, katerega red je enak 3. Naj bo $c \in \mathbb{Z}^*_N$ poljuben kubični ostanek, potem obstaja nek $x \in \mathbb{Z}^*_N$ za katerega velja $x^3 \bmod N = c$. Potemtakem so tudi $ax \bmod N$ in $a^2x \bmod N$ kubični koreni števila c. □

Opomba. Navedimo primer, ko imamo več kot tri kubične korene. Naj bo $N = 7 \cdot 13 = 91$ (potem je $\varphi(72) = 3^2 \cdot 2^3$) in $x^3 = 1$ kubični ostanek. Elementi množice $\{1, 9, 16, 22, 29, 53, 74, 79, 81\}$ predstavljajo kubične korene.

Prvi del protokola je namenjen skupnemu (nepristranskemu) vzorčenju vrednosti $x \in_R \mathbb{Z}_N^*$. To dosežemo z uporabo zapriseg. Drugi del protokola je izmenjevanje preverjevalčevega (Bojan) izziva (naključen kubični ostanek v $\mathbb{Z}_N^*$) in dokazovalčevega odgovora kubični koren izziva.

Uglašenost. Verjetnost, da je dokazovalec uspešen pri vseh izzivih ($\lceil \tau / \log_2(3) \rceil$ krogov) pri pogoju, da velja $3 \mid \varphi(N)$, je:

$$\left(\frac{\#\text{veljavni kubični koreni}}{\#\text{možni kubični koreni}} \right)^{\lceil \tau / \log_2(3) \rceil} \leq \left(\frac{1}{3} \right)^{\lceil \tau / \log_2(3) \rceil} \leq 2^{-\tau}.$$

Ker je bil x naključno (enakomerno) izbran, so vsi možni kubični koreni števila $x^3 \bmod N$ enako verjetni. Potemtakem preverjevalec sprejme neveljaven dokaz z verjetnostjo kvečjemu $2^{-\tau}$.

Polnost. Če je $3 \nmid \varphi(N)$, potem bo dokazovalec vedno vrnil pravi kubični koren (saj je enolično določen) in bo preverjevalec dokaz sprejel.

Dokaz brez razkritja znanja. Ker preverjevalec sodeluje z dokazovalecem pri "naključni" izbiri števila x, ne pridobi nobenega dodatnega znanja (v primeru poštenega dokazovalca). Od tod zaključimo, da gre pri tem protokolu res za dokaz brez razkritja znanja.

7.4 Protokol 4 – popačeno vezje

Skozi čas sta se razvili dve različici protokola "popačeno vezje" (Yaov protokol):

- Yaov protokol je odporen na pasivne napade (varnost slednjega protokola sta dokazala Lindell in Pinkas [26]).
- Različica odporna na aktivne napade (več različnih pristopov je bilo raziskanih, eden izmed njih je pristop cut-n-choose, glej [18, 24, 19]).

Za protokol 4 hočemo, da je odporen na aktivne napade. Za nas je pristop cut-n-choose neprimeren, ker je računsko preveč potraten. Poleg tega v našem primeru imamo zasebni vhod le s strani evalvatorja (vsi ostali argumenti so javni). Potemtakem lahko uporabimo osnoven Yaov protokol z naslednjimi dodatnimi varnostnimi zagotovili (glede aktivnih napadov):

- Zastrt prenos je oporen na aktivne napade. Varnost le tega se obravnava v naslednjem podpoglavju (7.5).
- Logično vezje C na kateremu se izvaja protokol 4 ni zlonamerno generirano. V našem primeru lahko zlonamernost vezja C preveri evalvator sam, ker ve kakšen bo rezultat funkcije F. To tudi stori v koraku (3d). S tem se izogne morebitnemu "uhajanju" svojih zasebnih vhodov ($K \,||\, m$) generatorju vezja.

Isti problem bi lahko rešili z zaprisegami. Evalvator zapriseže rezultat (izhodne oznake) generatorju. Ta mu odgovori z oznakama za vse možne logične vrednosti žic v popačenem vezju $\widetilde{C}$. S pridobljenimi oznakami preveri korektnost popačenega vezja $\widetilde{C}$. Če je bilo popačeno vezje korektno generirano, evalvator odpre oz. razkrije prej poslano zaprisego.

- Ali lahko evalvator priredi oznake izhodov, ker pozna vse vrednosti vhodov? Ne, Yaov protokol je varen, tudi v primeru da evalvator "ugane" generatorjev zaseben vhod, saj ima na vsakem koraku na voljo največ eno oznako (za eno izmed logičnih vrednosti) za vsako žico.

7.5 Protokol 5 – zastrt prenos

Navaden oz. osnoven Yaov protokol uporablja zastrt prenos, ki je odporen le na pasivne napade. Izkaže se, da v tem primeru lahko z aktivnimi napadi prejemnik pridobi obe pošiljateljevi sporočili (M_0 in M_1). Tega si mi ne moremo privoščiti, saj lahko evalvator potemtakem pri protokolu 4 ponaredi poljuben izhod. Nekatere izvedbe zastrtega prenosa (odpornega le na pasivne napade) "dopuščajo" aktivne napade s strani pošiljatelja, tj. z zlonamernim obnašanjem lahko pošiljatelj pridobi prejemnikov vhod (bit izbire c).

Opisan protokol 5 je odporen na aktivne napade [14], če izberemo aditivno grupo G, za katero je problem diskretnega logaritma težek. Za grupo G lahko izberemo grupo na točkah "zvite" Edwardsove krivulje

$$\left\{(x,y) \in \mathbb{F}_{2^{255}-19} \times \mathbb{F}_{2^{255}-19} \;\mid\; -x^2 + y^2 = 1 + dx^2y^2\right\}$$

z operacijo seštevanja

$$(x_1,y_1) + (x_2,y_2) = \left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 + x_1x_2}{1 - dx_1x_2y_1y_2}\right),$$

kjer je $d = -\frac{121665}{121666} \in \mathbb{F}_{2^{255}-19}$. Za generator grupe izberemo $B = (x, 4/5)$ s pozitivno koordinato x (točka je enolično določena).

Grupa G je izomorfna grupi $\mathbb{Z}_p \times \mathbb{Z}_8$, kjer je

$$p = 2^{252} + 27742317777372353535851937790883648493.$$

Varnostni parameter je v tem primeru $\approx 2^{128}$ ($\sqrt{\frac{\pi}{2}\mathrm{ord(B)}}$). Glavni razlog za izbiro te grupe je učinkovitost realizacije relevantnih aritmetičnih operacij, glej Chou in Orlandi [4].

7.6 Protokol 6 – poštena izmenjava ključev

Protokol 6 temelji na protokolu za pošteno izmenjavo RSA digitalnih podpisov, Boneh in Naor [7]. V našem primeru sta rezultata "podpisovanja sporočil" $E_{\mathrm{pub}_A}(K_A)$ in $E_{\mathrm{pub}_A}(K_A)$ z zasebnim RSA ključem podpisa oz. ključa K_A in K_B. Ker v takem načinu "podpisovanja" ne uporabljamo kriptografskih zgoščevalnih funkcij, lahko popisovalec (lažje) izda več različnih "podpisov" za isto "sporočilo". Zato je pomembno, da lastnik zasebnega ključa dokaže, da je (javni) šifrirni eksponent tuj z

njegovim javnim modulom. Ta problem smo odpravili med generiranjem in izmenjavo javnih ključev s protokolom 3.

Časovna prednost (v primeru predčasnega odstopa Ane ali Bojana) je skoraj izničena s korakom (1b). Brez tega koraka bi z zavlačevanjem protokola, lahko ob predčasnem odstopu Ane ali Bojana razlika v preostalem računskem delu bila velika, saj se lahko takoj, ko Ana pridobi Bojanovo izhodišče h_B, začne pripravljati na pridobitev ključa K_B s postopkom iz točke 3.

Poštenost pri pridobivanju ključa K z grobo silo je zagotovljena (tudi, če Ana ima večjo procesorsko moč, prek večjega števila procesorskih enot, kot Bojan). Trenutno se modularnega potenciranja namreč ne da dobro pralelizirat pri izračunu vrednosti v_i $(0 \leq i \leq \tau)$, tj. več procesorjev oz. jeder ne prinese nobene bistvene pohitritve, glej Fathy, Bahig in Ragab [17]. Poleg tega so razmerja med računskimi zmogljivostni za eno jedro za konvencionalne procesorje konstantna oz. zelo omejena.

7.7 Protokol 7 – dokaz brez razkritja znanja za Diffie-Hellmanovo četverko $\langle g, A, B, C\rangle$

Protokol temelji na klasičnem dokazu brez razkritja znanja za Diffie-Hellmanovo četverko $\langle g, A, B, C\rangle$.

Uglašenost. Naj bo q red elementa g_A v $\mathbb{Z}_{N_A}^*$ in naj bo d najmanjši praštevilski delitelj števila q. Če je Ana nepravilno konstruirala vektor W_A dolžine $\tau + 1$, je verjetnost, da bo v enem krogu dokaza ogoljufala Bojana največ

$$\tau\left(\frac{1}{\min(d,R)} + \frac{1}{R}\right),$$

kjer sta vrednosti τ in R varnostna parametra definirana v protokolu 6.

Dokaz slednjega lahko najdemo v Chaum in Pedersen [7, 13]. Tu glede na izbiro varnostnih konstant pri protokolu 6 ločimo dva primera:

- Varnostne konstante: $P = 3$, $R = 2^{90}$ in $Q = 1$. V tem primeru dokaz sestavlja samo en krog (tj. $Q = 1$). Najmanjši praštevilski delitelj d števila q je vsaj 2^{89}, ker je javni modul N_A produkt dveh varnih praštevil večjih od 2^{90}. Verjetnost uspešne goljufije je potemtakem kvečjemu

$$\tau\left(\frac{1}{\min(d,R)} + \frac{1}{R}\right) = 80\left(2^{-89} + 2^{-90}\right) \approx 2^{-82.1} \leq 2^{-\tau}.$$

- Varnostne konstante: $P = 128$, $R = 2^{10}$ in $Q = 13$. V tem primeru dokaz sestavlja 13 krogov (tj. $Q = 13$). Najmanjši praštevilski delitelj d števila q je vsaj $P = 128$, ker je $g_A = h_A^{\prod_{i=0}^{r} q_i^{\delta/2}} \bmod N_A$ in $h_A \in \mathbb{Z}_{N_A}^*$. Verjetnost uspešne goljufije je potemtakem kvečjemu

$$\tau\left(\frac{1}{\min(d,R)} + \frac{1}{R}\right)^{13} = 80\left(\frac{1}{128} + 2^{-10}\right)^{13} \approx 2^{-82.5} \leq 2^{-\tau}$$

Polnost. V koraku 5 Bojan preveri ali velja

$$g_A^{y_i} \cdot u_{i-1}^{-c_i} \equiv z_i \pmod{N_A} \quad \text{in} \quad u_{i-1}^{y_i} \cdot u_i^{-3c_i} \equiv w_i \pmod{N_A} \quad (1 \leq i \leq \tau).$$

Če razpišemo prvo kongruenco dobimo:

$$g_A^{y_i} \cdot u_{i-1}^{-c_i} \equiv g_A^{c_i \cdot 3 \cdot 2^{2^{i-1}} + \alpha_i} \left(g_A^{3 \cdot 2^{2^{i-1}}}\right)^{-c_i} \equiv g_A^{\alpha_i} \equiv z_i \pmod{N_A},$$

ker je $z_i = g_A^{\alpha_i} \bmod N_A$ in $y_i = c_i \cdot 3 \cdot 2^{2^{i-1}} + \alpha_i \bmod q$ ter $u_i = g_A^{3 \cdot 2^{2^i}} \bmod N_A$ $(0 \leq i \leq \tau)$. Pri drugi kongruenci pa

$$u_{i-1}^{y_i} \cdot u_i^{-3c_i} \equiv \left(g_A^{3 \cdot 2^{2^{i-1}}}\right)^{c_i \cdot 3 \cdot 2^{2^{i-1}} + \alpha_i} \left(g_A^{3 \cdot 2^{2^i}}\right)^{-3c_i} \equiv \left(g_A^{3 \cdot 2^{2^i}}\right)^{\alpha_i} \equiv w_i \pmod{N_A},$$

ker je $w_i = u_{i-1}^{\alpha_i} \bmod N_A$ in $y_i = c_i \cdot 3 \cdot 2^{2^{i-1}} + \alpha_i \bmod q$ ter $u_i = g_A^{3 \cdot 2^{2^i}} \bmod N_A$.

Protokol je **dokaz brez razkritja znanja** po Chaum in Pedersen [13]. Zaradi (precejšne) dolžine dokaza bralca vabimo, da si ga ogleda sam.

8 Ozadje implementacije, možne rešitve in optimizacije

Protokol 1 smo uspešno implementirali za GNU/Linux z naslednjimi omejitvami:

- Dolžini sporočil m_A in m_B sta večkratnika števila osem.
- Preverjamo lahko le naslednje specifikacije:
  - Ujemanje digitalnih odtisov za zgoščevalne funkcije MD5, SHA-1, SHA-256 za sporočila do dolžine 440 bitov oz. velikosti 55B.
  - Ujemanje digitalnih odtisov za zgoščevalno funkcijo SHA3-256 za sporočila do dolžine 4 096 000 bitov oz. velikosti 500KB.
- Brez uporabe varne povezave.

Pri implementaciji smo uporabili naslednje knjižnice, orodja in module:

- Python modul socket. Modul omogoča uporabo vmesnika za BSD vtičnice (angl. *sockets*).
- Python paket `PyCryptodome` (različica 3.8.2) [44]. Paket vsebuje večino osnovnih kriptografskih primitivov.
- Orodje `CBMC-GC` (različica 2.0) [11]. Orodje omogoča izdelavo optimiziranih logičnih vezij in podmnožice programov v programskem jeziku ANSI-C. Temelji na orodju `CBMC` [15] za formalno preverjanje programov v programskem jeziku ANSI-C.
- Orodje `emp-toolkit` [40]. Orodje ponuja kriptografske primitive za varno večstrankarsko računanje v programskem okolju C++.
- Logična vezja primerna za varno večstrankarsko računanje v "bristolski obliki" [38, 37].

Glavni protokol smo implementirali v programskem okolju Python 3 (različica 3.6.8). Komunikacijo med Ano in Bojanom smo implementirali z Python-ovi BSD vtičnicami. Protokol 4 smo implementirali v programskem okolju C++ z uporabo orodja `emp-toolkit`. Preveden program katerega funkcionalnost je protokol 4, smo integrirali v Python skripto glavnega protokola z uporabo Python modula subpocess. Paket `PyCryptodome` smo uporabili za osnovne kriptografske primitive kot so: generiranije praštevil za izdelavo RSA ključev, izračun kriptografskih zgoščevalnih funkcij (MD5, SHA-1, SHA-256 in SHA3-256), izračun simetrične šifre AES-128, računanje modularnega inverzna in največjega skupnega delitelja ter generiranje naključnih števil.

Tehnične implementacijske rešitve so naslednje:

- Za komunikacijo nismo uporabili varnostnega prenosnega sloja (angl. *secure socket layer*, kratica SSL), saj trenutno orodje `emp-toolkit` še ne podpira SSL.

- Lahko bi izmenjevali sporočila skoraj poljubne dolžine (omejeni smo z dolžino števca pri AES-128-CTR šifri), a zaradi slabšega upravljanja s pomnilnikom zmoremo izmenjevat sporočila le do dolžine 500KB. Z manjšimi spremembami bi lahko izmenjevali tudi daljša sporočila (samo za specifikacije, ki imajo konstantno prostorsko zahtevnost), a v tem primeru predstavlja problem omejena pasovna širina.

- Zaradi kompleksnosti protokola za dokaz brez razkritja znanja, da je javni modul N produkt dveh varnih vsaj ℓ-bitnih praštevil, smo pri preverjanju javnih ključev implementirali le protokol 3. Zato pri generiranju RSA ključev nismo uporabili varnih praštevil (ostale zahteve smo upoštevali), ampak smo konstruirani močni praštevili, tj. števila $p_1 - 1$, $p_1 + 1$, $p_2 - 1$ in $p_2 + 1$ imajo vsaj 100-bitne praštevilske faktorje. Četudi bi izbrali varni praštevili, bi pri protokolu 7 morali še vedno izvesti 13 krogov, saj nismo dokazali, da je javni modul N produkt dveh vsaj 90-bitnih varnih praštevil.

- Zaprisego pri protokolih 3 in 7 smo implementirali z zgoščevalno funkcijo SHA-256, tj. $\mathsf{Commit}(x, r) = \mathsf{SHA\text{-}256}(x \,||\, r)$.

- Pri protokolu 4 smo za dolžino oznak popačenega vezja uporabili $\kappa = 128$ namesto $\kappa + 1 = 129$. To smo storili zato, da lahko uporabimo simetrično šifro AES za funkcionalnost idealne zgoščevalne funkcije H. S tem smo si zmanjšali varnost varne evalvacije funkcije F (za en bit oz. za faktor dva), a smo pridobili na hitrosti evalviranja in generiranija popačenega vezja.

- Pri računanju komponent v_i vektorja Z_A Ana ne izračuna $v_i = g_A^{2^{2^i}} \bmod N_A$ neposredno, ampak najprej zmanjša eksponent $e_i = 2^{2^i} \bmod \lambda(N_A)$ in šele nato modularno potencira $v_i = g_a^{e_i} \bmod N_A$.

- Ker Ana pri protokolu 7 v splošnem ne pozna reda q elementa g_A, za q uporabi kar $\lambda(N_A)$, saj velja $q \,|\, \lambda(N)$.

- Ker Bojan pri protokolu 7 ne pozna $\lambda(N_A)$, izračuna $u_{i-1}^{-c_i} \bmod N_A$ kot

$$(u_{i-1}^{c_i})^{-1} \bmod N_A.$$

  Podobno velja tudi za $u_i^{-3c_i}$.

Ostale implementacijske rešitve in optimizacije bomo obravnavali v naslednjih podpoglavjih.

8.1 Razširitev zastrtega prenosa odpornega na aktivne napade

Pri protokolu 5 potrebujemo za vsak bit evalvatorjeva argumenta (vhoda) funkcije F en zastrt prenos. Ker zastrt prenos temelji na asimetrični kriptografiji oz. kriptografiji javnih ključev, je potrebno opraviti veliko računskega dela in obdelati veliko komunikacije med prejemnikom in pošiljateljem.

Poglejmo si primer za sporočilo m velikosti 1MB, s ključem K velikosti 128B (evalvatorjev vhod v protokol 4). Za vsak bit sporočila m in ključa m moremo pridobiti pripadajočo 128-bitno oznako $w_i^{0/1}$ za vhodno žico w_i vezja C. Če uporabimo protokol 5, imamo pri posamezni izvedbi protokola (za vsak bit sporočila m in ključa K) naslednjo količino podatkov komunikacije:

- Za predstavitev elementov (S in R) grupe G potrebujemo 768 bitov, Chou in Orlandi [14].
- Šifrirani sporočili e_0 in e_1 dolžine 128 bitov, ki predstavljata šifrirani oznaki w_i^0 in w_i^1.

Torej moramo za vsako izvedbo prenesti vsaj 1024 bitov. To pomeni, da za izbrano sporočilo m in ključ K potrebujemo vsaj 1GB podatkov medsebojne komunikacije. Nastali problem lahko rešimo s t.i. razširitvami zastrtega prenosa (angl. *oblivious transfer extension*, kratica OTe), glej Asharov, Lindell et al. [1]). Analogno mešanemu načinu šifriranja daljših sporočil pri kriptografiji javnih ključev (tj. sporočilo zašifriramo s simetrično šifro, pri tem uporabljen ključ pa z javnim ključem prejemnika), lahko z uporabo simetričnih kriptografskih primitivov manjše število osnovnih zastrtih prenosov razširimo na polinomsko več. Tak primer razširjenega zastrtega prenosa je Keller, Orsini in Scholl [22]. Če ga uporabimo (s 128-bitnimi računskimi in 40-bitnimi statističnimi varnostnimi parametri) za prejšnji problem večjega sporočila m s ključem K, potrebujemo naslednjo količino podatkov komunikacije:

- "priprava" potrebuje 128 osnovnih zastrtih prenosov, ki skupaj znaša 80Kb podatkov komunikacije,
- za vsak bit sporočila in ključa se prenese le 128 bitov.

To pomeni, da za izbrano sporočilo m in ključ K potrebujemo približno 128MB podatkov medsebojne komunikacije, kar je osem-krat manj, kot če uporabimo osnoven (nerazširjen) zastrt prenos. Poleg tega imamo v primerjavi s prejšnjim protokolom manj računskega dela.

V obeh primerih (navaden in razširjen zastrt prenos) gre za protokola katere realizacija spada v kategorijo najučinkovitejših (dobro izkoriščata arhitekturo novejših CISC procesorjev). Pri realni implementaciji smo uporabili razširitev zastrtega prenosa iz Keller, Orsini in Scholl [22], z $\mathcal{O}(\kappa)$-bitno računsko ($\kappa = 128$) in $\mathcal{O}(s)$-bitno statistično varnostjo ($s = 64$), ki je odporna na aktivne napade s strani pošiljatelja, kot tudi s strani prejemnika. Razširitev tega zatrtega prenosa smo implementirali z orodjem `emp-toolkit`.

8.2 Optimizacije popačenega vezja

Pri konstrukciji logičnih vezij bomo uporabili le logična vrata AND, XOR in NOT. Ta omejitev nam omogoča naslednje optimizacije. Naslednje metode so za vrata z dvema vhodoma. Vse spodaj opisane metode optimizacije so uporabljene pri implementaciji protokola 4 – popačeno vezje z orodjem `emp-toolkit`.

Pokaži in premešaj (angl. *point and permute*) [36]. Ker v splošnem ne vemo katera vrednost v šifrirani resničnostni tabeli za neka vrata v protokolu "popačeno vezje" predstavlja pravi izhod, je potrebno (v najslabšem primeru) poskusiti odšifrirat vse štiri vrednosti (za dvo-vhodna vrata). Pravi izhod prepoznamo po ničlah na koncu čistopisa (te ničle so bile dodane pri generiranju popačenega vezja). Temu se lahko izognemo z uporabo zadnjih bitov vhodnih oznak, ki enolično določajo vrstico v tabeli, ki predstavlja šifriran izhod pri danih vhodnih oznakah.

Popačena redukcija vrstic (angl. *garbled row reduction*) [30]. Če fiksiramo enega izmed šifriranih izhodov, nam tega ni potrebno poslati evalvatorju. Potem takem je evalvatorju potrebno poslati le tri oznake. Eno izmed dveh izhodnih oznak lahko izberemo na tak način, da bo zadnja šifrirana oznaka v tabeli enaka bitnemu nizu ničel.

Brezplačna-XOR-vrata (angl. *free-XOR*) [23]. Z manjšo spremembo pri generiranju oznak lahko število šifriranih možnih izhodov za logična vrata XOR zmanjšamo na nič. Poleg tega pri evalvaciji popačenih vrat XOR ni potrebno izvesti nobenega odšifriranja. To metodo implementiramo tako, da pri protokolu 4 zamenjamo generiranje popačenega vezja z naslednjim postopkom:

1. Generator izbere globalen diferenčni ključ $\Delta \in_R \{0,1\}^\kappa$.
2. Generator za vsako vhodno žico w_i vezja C izbere oznako w_i^0 za logično vrednost nič:
$$w_i^0 = (k_i^0, p_i^0) \in_R \{0,1\}^{\kappa+1}$$
in izračuna oznako w_i^1 za logično vrednost ena:
$$w_i^1 = (k_i^1, p_i^1) := (k_i^0 \oplus \Delta, p_i^0 \oplus 1).$$
3. Za vsaka logična vrata G_i v topološkem vrstnem redu vezja C (od vhodov proti izhodom) z vhodnimi oznakami
$$w_a^0 := (k_a^0, p_a^0),\ w_a^1 := (k_a^1, p_a^1),\ w_b^0 := (k_b^0, p_b^0),\ w_b^1 := (k_b^1, p_b^1),$$
glede na vrsto vrat naredi naslednje:
   (a) Naj bodo G_i logična vrata, ki implementirajo funkcijo $w_c = \mathsf{XOR}(w_a, w_b)$, potem generator izračuna izhodne oznake kot
   $$w_c^0 := (k_a^0 \oplus k_b^0, p_a^0 \oplus p_b^0),\ w_c^1 := (k_a^0 \oplus k_b^0 \oplus \Delta, p_a^0 \oplus p_b^0 \oplus 1).$$
   (b) Za vsa druga G_i logična vrata z dvema vhodoma ki implementirajo funkcijo $w_c = g(w_a, w_b)$, generator izbere oz. izračuna izhodne oznake kot
   $$w_c^0 = (k_c^0, p_c^0) \in_R \{0,1\}^{\kappa+1},\ w_c^1 := (k_c^0 \oplus \Delta, p_c^0 \oplus 1)$$
   ter generira popačeno tabelo za vrata G_i tako, da za vse 4 možne kombinacije vhodnih vrednosti $v_a, v_b \in \{0,1\}$ vrat G_i izračuna
   $$e_{v_a,v_b} := H(k_a^{v_a} \mid\mid k_b^{v_b} \mid\mid i) \oplus w_c^{g_i(v_a,v_b)},$$
   kjer je $H : \{0,1\}^* \to \{0,1\}^{\kappa+1}$ idealna zgoščevalna funkcija. Vrednosti e_{v_a,v_b} vstavi v tabelo in jih uredi po ključu $(p_a^{v_a}, p_b^{v_b})$.

4. Postopek za izračun izhodnih tabel ostane nespremenjen.

Popraviti moramo tudi evalvacijo popačenega vezja za logična vrata XOR. Zanje po metodi brezplačna-XOR-vrata nimamo popačene tabele. Izhodno oznako w_c, pri vhodnih oznakah $w_a = (k_a, p_a)$ in $w_b = (k_b, p_b)$, evalvator izračuna neposredno na naslednji način:

$$w_c = (k_a \oplus k_b, p_a \oplus p_b).$$

Pol-vrata (angl. *half gates*) [43]. Metoda omogoča evalvacijo popačenih vrat AND s samo dvemi šifriranimi izhodi, pri čemer ohranimo združljivost z metodo brezplačna-XOR-vrata. Metoda je optimalna za nelinearna vrata z dvema vhodoma. Za evalvacijo vrat AND uporabimo evalvatorjeva "pol-vrata", generatorjeva "pol-vrata" in vrata XOR. Pol-vrata so vrata, kjer je eden izmed vhodov znan. Vrata AND evalviramo tako, da jih najprej razdelimo na generatorjeva "pol-AND vrata" in na evalvatorjeva "pol-AND vrata" ter jih na koncu združimo prek XOR vrat. Za vsaka "pol-AND vrata" generator in evalvator sodelujeta pri skupnem izračunu. Evalviranje, generiranje in združevanje generatorjeva in evalvatorjevih "pol-AND vrata" izvajamo na naslednji način:

- Generatorjeva "pol-AND vrata". Namen teh vrat je izračunati $v_c = v_a \wedge v_b$, kjer je vrednost v_a generatorju znana. Generator glede na vrednost v_a izračuna naslednjo popačeno tabelo za vhodni oznaki b^0 in b^1 (za vhod v_b) ter izhodne oznake c^0 in $c^1 = c_0 \oplus \Delta$ (za izhod v_c) z naslednjimi elementi na način brezplačnih XOR vrat:

  $$H(b^0) \oplus c^0 \quad \text{in} \quad H(b^1) \oplus c^0 \oplus v_a \cdot \Delta.$$

  Tabela je urejena po metodi "pokaži in premešaj" ter zmanjšana na eno samo eksplicitno šifrirano oznako prek metode popačena redukcija vrstic. Evalvator lahko iz take table pridobi končen rezultat (rezultatu pripadajočo izhodno oznako), če pozna eno izmed vhodnih oznak b^0 in b^1.

- Evalvatorjeva "pol-AND vrata". Namen teh vrat je izračunati $v_c = v_a \wedge v_b$, kjer sta v_a in pripadajoča oznaka a^{v_a} evalvatorju znana. Od generatorja evalvator prejme naslednja elementa popačene tabele za vhodni oznaki a^0 in a^1 na način brezplačnih XOR vrat:

  $$H(a^0) \oplus c^0 \quad \text{in} \quad H(a^1) \oplus c^0 \oplus b^0.$$

  Tu ne uporabimo metode "pokaži in premešaj", saj evalvator že pozna vrednost v_a. Še vedno pa lahko uporabimo metodo popačena redukcija vrstic. V primeru da vrednost oznake v_a predstavlja logično vrednost nič, evalvator pridobi izhodno oznako $c^0 = (H(a^0) \oplus c^0) \oplus H(a^0)$, drugače pa oznako

  $$c^0 \oplus b^0 = (H(a^1) \oplus c^0 \oplus b^0) \oplus H(a^1).$$

  Iz te oznake lahko izračuna izhodno $c^{v_b} = (c^0 \oplus b^0) \oplus b^{v_b}$. To velja, ker je $c^1 = c^0 \oplus \Delta$ in $b^1 = b^0 \oplus \Delta$.

- Združevanje "pol-vrat". Evalviranje navadnih AND vrat $v_c = v_a \wedge v_b$ lahko prevedemo na evalvacijo dveh pol-AND vrat in XOR vrat tako, da generator "izbere" naključen bit $r = p_a^0$, vrata AND pa izrazi na naslednji način:

$$v_c = v_a \wedge v_b = (r \oplus r \oplus v_a) \wedge v_b = (r \wedge v_b) \oplus ((r \oplus v_a) \wedge v_b).$$

  Vrata $r \wedge v_b$ so generatorjeva (le-ta pozna r), vrata $(r \oplus v_a) \wedge v_b$ pa evalvatorjeva (slednji pozna vrednost $r \oplus v_a$, saj velja $r \oplus v_a = p_a^{v_a}$, ker je $p_a^1 = 1 \oplus p_a^0$). Izhodni oznaki posameznih vrat združimo prek XOR vrat in s tem pridobimo končni rezultat (izhodno oznako c^{v_c}).

Uporaba cevovoda (angl. *pipe-lining*) [20]. Generiranije in evalviranje popačenega vezja izvajamo "istočasno". Ta metoda je potrebna za uporabo metode half-gates. Z njeno uporabo lahko evalviramo in generiramo večja vezja, četudi imamo na voljo le omejeno količino pomnilnika, saj oznake žic in popačene tabele pošiljamo oz. prejemamo sproti. Poleg tega lahko med samim izvajanjem protokola oznake, ki jih ne potrebujemo več, odstranimo iz pomnilnika. Prednost je tudi v tem, da sta oba udeleženca več čas aktivna (udeleženca se ne čakata veliko).

Strojna podpora. Z uporabo Intelovih naborov ukazov AES-IN lahko učinkovito implementiramo šifriranje in odšifriranje oznak z uporabo šifre AES. Za operacije nad oznakami uporabimo 128-bitne registre, ki so del razširitve nabora ukazov – Streaming SIMD Extensions 2.

Vrata OR in NOT. Vrata OR lahko zamenjamo z vrati AND in NOT prek De Morganovega zakona ($\mathsf{OR}(v_a, v_b) = \mathsf{NOT}(\mathsf{AND}(\mathsf{NOT}(v_a), \mathsf{NOT}(v_b)))$). Vrata NOT enostavno implementiramo tako, da uporabimo tabelo z dvema elementoma (oz. enim, če fiksiramo eno šifrirano oznako).

8.3 Modularnost popačenega vezja

V koraku (2a) protokola 1 Ana in Bojan želita varno izračunati funkcijo F_A nad zasebnim vhodom Ane. Funkcija F_A ima le en argument, saj so argumenti s strani Bojana javni. Bojan svoje argumente uporabi pri pripravljanju logičnega vezja C za funkcijo F_A. Ta pristop je neprimeren za implementacijo, saj si ne moremo pripravit logičnih vezij vnaprej. Nastali problem rešujemo tako, da javne podatke vnese generator prek svojega zasebnega vhoda, tj. dodaten argument funkcije F_A, ki je namenjen generatorju. Dodaten generatorjev vhod implementiramo tako, da v koraku (3) protokola 4 generator pošlje za vsako bit svojega privatnega vhoda evalvatorju pripadajočo vhodno oznako. Zdaj si lahko generator vnaprej pripravi univerzalna vezja, ki jih utegne uporabiti za med seboj podobne specifikacije (npr. preverjanje različnih digitalnih odtisov za iste zgoščevalne funkcije pri enakih dolžinah sporočil).

Oba pristopa uporabljata logična vezja v monolitni obliki. Velikost takega vezja je hitro neobvladljiva, četudi bi uspeli vezje na nek način optimizirat. Za sporočilo velikosti 500KB (za preverjanje zgoščenih vrednosti popularnih zgoščevalnih funkciji), bi potrebovali vezje z več kot 10^9 logičnimi vrati že samo za implementacijo izračuna zgostitve in šifriranja sporočila. Izmenjava takih vezij je časovno potra-

tna (tudi, če uporabimo metode za stiskanje vezij), optimizacija pa v realnem času nedosegljiva.

Zaradi lepe odvisnosti med gradniki funkcije F_A lahko vezje razdelimo na manjše dele oz. manjša logična vezja. Iz teh delov lahko hitro sestavimo celotno vezje samo z dodajanjem novih žic oz. povezav. Pri povezovanju moremo paziti le, da ima sestavljeno vezje topološko ureditev vrat. Take manjše gradnike lahko dobro optimiziramo. Tako sestavljeno vezje ni potrebno dodatno optimizirati. Intuitivno bo tako vezje zelo blizu rezultatu globalne optimizacije, saj posamezni gradniki prestavljajo kriptografske primitive. Če bi bila razlika med globalno in lokalno optimizacijo velika, bi to pomenilo, da uporabljamo manj varne kriptografske primitive.

V našem primeru so taki gradniki:

- kompresijske funkcije zgoščevalnih funkcij MD5, SHA-1 in SHA-256,
- "permutacijska" funkcija KECCAK-f, ki je osnova za SHA3-256,
- bločna šifra AES-128,
- množenje in računanje ostanka za 1024 in več-bitna števila ter
- logična operacija XOR.

Preostane nam še obravnava problema optimizacije gradnikov. Orodja in izdelana logična vezja za te gradnike že obstajajo, vendar so cilji optimizacije namenjeni za razvoju integralnih vezij (npr. integrirana vezja ASIC za iskaje dokazila o delu pri kriptovalutah). V našem primeru želimo le, da ima vezje majhno število vrat, pri pogoju, da vezje vsebuje samo vrata AND, NOT in XOR. V kontrastu z optimizacijo pri sintezi logičnih vezij namenjenih za integrirana vezja nas pri optimizaciji gradnikov ne zanima kako so vrata med seboj povezana. Poleg tega taka orodja dajo ponavadi prednost vratom NAND (cenejša izvedba), mi pa želimo, da je delež vrat XOR čim večji (izkoriščanje metode brezplačna-XOR-vrata), a ne na račun bistveno večjega števila vrat. Z manjšimi spremembami lahko logična vezja ki jih pridobimo z uporabimo orodij za sintezo logičnih vezij namenjenih izdelavi integriranih, še vedno uporabimo (stopnja optimizacije je še vedno primerna). Problem je le v tem, da ta orodja niso prosto dostopna.

Vezja gradnikov smo pridobili na dva načina:

- Uporaba orodja `CBMC-GC`. Omogoča izdelavo vezij namenjenih za varno večstrankarsko računanje iz skoraj poljubnega programa v jeziku ANSI-C. Problem je le v tem, da orodje potrebuje veliko pomnilnika (več kot 16GB) za večja vezja (več 10 milijonov vrat). Če želimo, da bo izdano vezje dovolj dobro optimizirano, je čas optimizacije velik (več dni za vezja z več milijoni vrat).
- Sprememba že izdelanih logičnih vezij (za funkcije MD5, SHA-1, SHA-256, KECCAK-f in AES-128) kriptografske raziskovalne skupine univerze v Bristolu. Ta vezja so bila izdelana z uporabo orodij za sintezo digitalnih vezij.

Omejili smo se le na množico specifikacij, ki so zasnovana na popularnih zgoščevalnih funkcijah. Z uporabo orodja `CBMC-GC` bi lahko implementirali preverjanje poljubnih specifikacij, če le ni pripadajoče vezje preveliko. Načeloma je čas evalvacije funkcij prek protokola "popačeno vezje" za konstantni faktor počasnejša od neposredne evalvacije, če le nimamo veliko opravka z RAM-om. Za specifikacije katere preverjanje potrebuje veliko dostopov do RAM-a, je smiselneje uporabiti (manj varne) zastrte podatkovne strukture (angl. *oblivious data structures*), glej [16, §5]. Le-te so le ena možna razširitev za protokol "popačeno vezje", zato pripadajočih protokolov nismo raziskovali.

8.4 Kriptografske zgoščevalne funkcije prek popačenih vezij

Zgostitev za zgoščevalne funkcije MD5, SHA-1 in SHA-256 nekega sporočila m izračunamo na naslednji način:

- Najprej sporočilu m dodamo bitno zapolnitev (angl. *padding*) $1 \,||\, 0^j \,||\, \ell$, kjer je 0^j niz j-tih ničel in ℓ dolžina sporočila m po modulu 2^{64} v binarnem zapisu. Vrednost j je najmanjše nenegativno število, ki ustreza enačbi
$$|m| + 1 + j + 64 \equiv 0 \pmod{512}.$$
Tako smo dobili sporočilo $m' = m \,||\, 1 \,||\, 0^j \,||\, \ell$.

- Za vsak 512-biten blok $m'_i = m'[128i : 128(i+1)]$ sporočila m' izračunamo
$$w_i = f(w_{i-1}, m'_i) \quad (1 \leq i \leq |m'|/512),$$
kjer je w_i vektor delavnih spremenljivk (angl. *working variables*) v i-ti iteraciji in f kompresijska funkcija, ki pripada neki zgoščevalni funkciji $H \in \{\mathsf{MD5}, \mathsf{SHA\text{-}1}, \mathsf{SHA\text{-}256}\}$. Vektor w_0 predstavlja inicializacijo, ki je določena za vsako zgoščevalno funkcijo posebej. Dolžina vektorjev w_i je 128 za MD5, 160 za SHA-1 in 256 za SHA-256.

- Zgostitev je $H(m) = w_{|m'|/512}$.

Če imamo vezje, ki implementira kompresijsko funkcijo f za pripadajočo zgoščevalno funkcijo, lahko izdelamo pripadajoče logično vezje (s kaskadno vezavo) za sporočila poljubne dolžine, saj bitno zapolnitev lahko generator vpelje v protokol, kar prek svojega zasebnega vhoda. Ker je odvisnost gradnikov zaporedna, lahko vezje sestavljamo medtem ko, generiramo popačeno vezje.

Za kompresijske funkcije smo z manjšimi spremembami uporabili logična vezja za kompresijske funkcije MD5, SHA-1 in SHA-256 univerze v Bristolu. Ker ta vezja pripadajo kompresijskim funkcijam s fiksiranimi delovnimi spremenljivkami na ustrezen inicializacijski vektor, jih lahko uporabimo le za sporočila, ki so krajša od 447 bitov. Vezja smo uspeli pripravit tudi z orodjem `CBMC-GC`, a so bila slabše optimizirana, ker smo bili časovno omejeni pri izvajanju optimizacije.

Zgostitev za zgoščevalno funkcijo SHA3-256 nekega sporočila m izračunamo na naslednji način:

- Vsakemu bajtu sporočila m se zrcali pripadajoče bite (vrstni red bitov za posamezen bajt se zamenja).

- Tako spremenjenemu sporočilu m dodamo bitno zapolnitev (angl. *padding*) $011 \,||\, 0^j \,||\, 1$, kjer je 0^j niz j-tih ničel. Vrednost j je $(-|m| - 4)$ mod 1088. Tako smo dobili sporočilo $m' = m \,||\, 011 \,||\, 0^j \,||\, 1$.

- Za vsak 1088-biten blok $m'_i = m'[1088i : 1088(i+1)]$ sporočila m' izračunamo

$$S_i = \mathsf{KECCAK\text{-}f}(S_{i-1} \oplus (m'_i \,||\, 0^{512})) \quad (1 \leq i \leq |m'|/1088),$$

  kjer je $S_0 = 0^{1600}$ in KECCAK-f "permutacijska" funkcija

$$\mathsf{KECCAK\text{-}f} : \{0,1\}^{1600} \to \{0,1\}^{1600}.$$

- Zgostitev je $\mathsf{SHA3\text{-}256}(m) = S_{|m'|/1088}[0 : 256]$.

Na podoben način kot pri prejšnjih primerih zgoščevalnih funkcij lahko tudi tu uporabimo način povezovanja logičnih vezij, ki predstavljajo funkcijo KECCAK-f, v večje vezje s funkcionalnostjo zgoščevalne funkcije SHA3-256. Na vsakem koraku moramo dodat še vezje, ki implementira operacijo $\oplus$ med trenutnim blokom m'_i in trenutnim stanjem S_i. Za vsak korak potrebujemo le 1088 dodatnih XOR vrat. Z minimalnimi spremembami implementacije za preverjanje zgostitve za zgoščevalno funkcijo SHA3-256, bi lahko omogočili preverjanje zgostitev za zgoščevalne funkcije SHA3-224, SHA3-384, SHA-512, SHAKE128 in SHAKE256.

Namesto, da v popačenem vezju primerjamo ali se podana zgostitev H_m ujema z zgostitvijo $H(m)$ evalvatorjevega sporočila m prek popačenega vezja izračunamo kar $H_m \oplus H(m)$. Generator bo sprejel verifikacijo specifikacije, če bo $H_m \oplus H(m)$ niz ničel. Logična vezja, ki pripadajo verifikacijski funkciji (v tem primeru preverjanju zgoščenih vrednosti), integriramo enostavno v prestali del logičnega vezja tako za vezje neposredno povežemo z vhodom evalvatorjevega sporočila m ter na pomožne vhode generatorja. Pri implementaciji zgoščevalnih funkcij smo se držali standardov FIPS 180 [32] in FIPS 202 [33].

8.5 AES prek popačenih vezij

Sporočilo šifriramo s šifro AES-128 v načinu delovanja števec. Za implementacijo njene funkcionalnosti smo za bločno šifro AES-128 uporabili logična vezja univerze v Bristolu. To vezje smo uspešno izdelali tudi prek orodja `CBMC-GC` ampak s približno 6-krat več vrati. Zaradi lažje implementacije nismo integrirali povečevanje števca v logičnem vezju, ampak za posamezen blok sporočila vrednost števca prenesemo kar prek generatorjeva vhoda v protokolu "popačeno vezje". Tudi inicializacijski vektor vpeljemo prek generatorjeva vhoda, a le enkrat. S tem je šifriranje blokov medsebojno neodvisno in je zlaganje pripadajočih logičnih vezij enostavno. Potrebujemo namreč le povezati izhode logičnih vezij bločne šifre AES-128 z vhodi, ki so namenjeni evalvatorjevemu zasebnemu sporočilu prek logičnih vrat XOR. Evalvator zagotovi, da sporočilo smiselno dopolnjeno z naključnimi biti.

8.6 RSA prek popačenih vezij

Za krajša sporočila izračun $K^3 \bmod N$ prek popačenih vezij najbolj potratna operacija, kljub temu, da smo namensko izbrali najmanjši možen šifrirni eksponent. Analizirali oz. obravnavali smo naslednje načine računanja $K^3 \bmod N$:

- Izračun $K^3 \bmod N$ kot $((K \cdot K) \bmod N) \cdot K \bmod N$. V tem primeru potrebujemo dve množenji nad 1024-bitnimi števili in dve deljenji z ostankom pri 2048-bitnemu deljencu in 1024-bitnemu delitelju.

- Izračun $K^3 \bmod N$ kot $(K \cdot K \cdot K) \bmod N$. V tem primeru potrebujemo eno množenje nad 1024-bitnimi števili, eno množenje s 1024 in 2048-bitnimi faktorji ter eno deljenje z ostankom za 3072-bitni deljenec in 1024-bitni delitelj. V teoriji bi mola biti ta metoda boljša od prve vendar se v praksi izkaze za slabšo. Vzrok je slabša optimizacija zaradi večjega števila logičnih vrat v pripadajočih vezij za množenje in računanje deljenja z ostankom.

- Evalvator bi lahko generatorju poslal zadnji del zasebnega ključa $K[128 : 1024]$. Prek popačenega bi bilo potrebno evalvirati le

$$a(K[0:128])^3 + b(K[0:128])^2 + c(K[0:128]) + d \bmod N,$$

  kjer je $a = 2^{3 \cdot 896} \bmod N$, $b = 2^{2 \cdot 896} K[128 : 1024] \bmod N$, $c = 2^{896}(K[128 : 1024])^2 \bmod N$ in $d = (K[128 : 1024])^3 \bmod N$. Vrednosti a, b, c, in d bi generator vpeljal v protokol "popačeno vezje" prek svojega zasebnega vhoda. Na ta način bi pohitrili izvajane šifriranja za približno faktor 4. Za tako razkritje nismo uspeli najti napada, to pa še ne pomeni, da je varen način izračuna.

Logična vezja za množenje in deljenje z ostankom smo konstruirali z orodjem `CBMC-GC`. Za množenje in deljenje z ostankom smo uporabili šolsko metodo, ki ima časovno zahtevnost $\mathcal{O}(n^2)$. Za množenje bi lahko uporabili Karatsubov algoritem. Ker ima časovno zahtevnost $\mathcal{O}(n^{\log_2 3})$ bi pri 1024-bitnih številih dosegli približno 17-kratno pohitritev. Pri deljenju bi lahko uporabil algoritem "deli in vladaj deljenje" (angl. *divide and conquer division*), glej [10, §1.4.3]. Montgomeryjevo modularno množenje [29] v našem primeru pride v poštev, ker potrebujemo le dve modularni množenji in bi bila pretvorba v Montgomeryjevo obliko preveč zamudna in draga.

9 Alternativni protokoli in pristopi

Poleg naše rešitve bi lahko izmenjavo sporočil izvedli z naslednjimi pristopi:

- Uporaba strojnih nepropustnih naprav (angl. *hardware security module*), glej Avoine, Guerraoui et al. [2]. V tem primeru oba udeleženca potrebujeta ustrezno nepropustno napravo.

- Ključa K_A in K_B bi lahko izmenjali prek razkrivanja posameznih bitov. To bi storili tako, da ključev K_A in K_B ne šifriramo, ampak si udeleženca postopno razkrivata oznake, ki pripadajo vhodom ključev popačenega vezja. Ker generator pozna vse oznake žic za se možne vhode, lahko ob vsaki izmenjavi preveri ali je prejeta oznaka veljavna oz. ali je prejeti bit res del ključa. Taka izmenjava je hitra, a smo s tem izpostavljeni vzporednim napadom. To pomeni, da je napad izčrpnega preiskovanja za neprejete bite ključa možno pralelizirat. V primeru da ima en izmed udeležencev večjo paralelno moč, je le-ta lahko na boljšem za poljuben faktor.

- V protokolu 6 bi lahko razmerje med računskim delom potrebnim za pridobitev ključev K_A in K_B med udeležencema v primeru predčasnega odstopa zmanjšali iz dva na poljuben faktor (npr. na faktor $\alpha > 1$). Za doseganje varnostnega parametra $T = 2^\tau$ bi potrebovali vektorje W in Z dolžine $\lceil \tau \log_\alpha 2 + 1 \rceil$ in temu primerno število korakov pri postopni izmenjave elementov vektorjev Z. Če bi za vektor Z izbrali

  $$Z = \left\langle g^{2^{f_0}}, g^{2^{f_2}}, g^{2^{f_3}}, ..., g^{2^{f_{\tau'}}} \right\rangle \bmod N,$$

  kjer je $f_{-1} = 0$, $f_0 = 1$, $f_i = f_{i+1} + f_{i+2}$ (tj. zamaknjeno Fibonaccijevo zaporedje), je faktor $\alpha \approx 1.62$. Za doseganje varnostnega parametra $T = 2^\tau = 2^{80}$ je potrebno pri taki konstrukciji izbrati $\tau' = 116$. Ustrezno je treba tudi popraviti konstrukcijo vektorja W iz protokola 7 za preverjanje korektnosti konstitucije vektorja W.

- Pri šifriranju ključev K_A oz. K_B bi lahko uporabili druge kriptosisteme iz kriptografije javnih ključev, če bi le znali ustrezno spremeniti protokol 6. Šifriranje s kriptosistemom RSA pri šifrirnem eksponentu 3 se da najučinkoviteje izračunat prek protokola "popačeno vezje". Ostali znani kriptosistemi z javnimi ključi so namreč počasnejši, čeprav ima kriptosistem RSA daljše ključe.

- Če imamo na voljo na pol pošteno tretjo osebo lahko izmenjavo ključev izvedemo prek le-te. Preverjanje specifikacij, šifriranje ključev in sporočil prek protokola "popačeno vezje" je še vedno potrebno. Ta oseba sprejme ključa K_A in K_B ter pripadajoča šifrirana ključa $E_{\text{pub}_A}(K_A)$ in $E_{\text{pub}_B}(K_B)$. Ključa posreduje (izmenja) Ani in Bojanu, če prejeta ključa K_A in K_B ustrezata prejetim šifriranim ključem $E_{\text{pub}_A}(K_A)$ in $E_{\text{pub}_B}(K_B)$. Ker tretja oseba ne pozna pripadajočih šifriranih sporočil $K_A(m_A)$ in $K_B(m_B)$, s prejetimi ključi ne pridobi nobene nove koristne informacije.

- Pošteno menjavo sporočil bi bilo mogoče implementirati z uporabo platforme Ethereum [41]. Problem takšne implementacije je, da lahko izmenjujemo le krajša sporočila. V tem primeru bi za zagotavljanje atomarnosti potrebovali denar v kriptovalutah (podobno kot v primeru izvajanja transakcij prek protokola Lighting network [34]).

10 Testi zmogljivosti

Implementacijo protokola 1 smo za vse implementirane specifikacije testirali v naslednjih okoliščinah:

- povratna zanka (angl. *loopack*), tj. računalnik komunicira sam s seboj,
- dva računalnika povezana v lokalni mreži s hitrostjo povezave 100Mb/s (način full duplex) s pingom < 1ms.
- dva računalnika povezana v lokalni mreži s hitrostjo povezave 1Gb/s (način full dubplex) s pingom < 0.5ms.

Testirali smo specifikacije za ujemanje digitalnih odtisov za zgoščevalne funkcije MD5, SHA-1, SHA-256 za sporočila velikosti 55B. Za zgoščevalno funkcijo SHA3-256 smo primerjali zmogljivost implementiranega protokola pri različnih velikostih sporočil (od 55B do 500KB). Vse meritve smo izvajali tako, da smo od 4 krat do 30 krat (odvisno od dolžine izvajanja protokola) izvedli protokol postopne poštene izmenjave sporočil. Pri tem smo izmenjevali naključni sporočili enake dolžine pri enakih specifikacijah. Brez težav bi lahko izmenjevali sporočila pri različnih specifikacijah, a bi rezultate meritev težko primerjali.

Meritve smo izvajali na dveh računalniki z operacijskim sistemom ArchLinux (različica jedra 5.2.9). Računalnik, ki smo ga uporabili za meritve s povratno zanko, je imel procesor Intel i7-3930K, drugi računalnik pa Intel i7-4600U. V tabelah 1, 2, 3, 4 in 5 je poleg časa izvajanja posameznega procesa podana standardna napaka.

V tabelah 1 in 2 so meritve trajanja posameznih delov protokola, ki niso odvisni od specifikacij. Opazimo, da pasovna širina ne vpliva na trajanje izvajanja preverjanja javnega modula ter na postopno izmenjavo ključev, saj pri tem izmenjujemo le majhno količino podatkov (manj kot 1MB). Odstopanja bi opazili šele, če bi bila računalnika na veliki razdalji oz. bi bila povratna zakasnitev (angl. *round-trip delay time*) velika. V tabeli 6 razberemo, da količina prenesenih podatkov narašča linearno z velikostjo izmenjanih sporočil. To ni presenetljivo, saj je časovna zahtevnost šifriranja podatkov s simetrično šifro in računanje zgoščevalne funkcije SHA3-256 linearna. Iz tabel 3, 4, 5 in 6 je razvidno, da je čas izvajanja in količina prenesenih podatkov približno enaka (za isto pasovno širino) za primere izmenjave krajših sporočil. Vzrok za to je računsko potratna operacija množenja in računanja ostanka za 2048 bitna števila pri šifriranju ključa K.

Če primerjamo trajaje protokolov pri pasovni širini 100Mb/s s pasovno širino 1Gb/s, je faktor pohitritve približno 10. Tako implementiran protokol 1 je smiseln za izmenjavo sporočil do dolžine 50KB, če imamo na voljo povezavo s hitrostjo 100Mb/s (v obe smeri) in do dolžine 500KB, če imamo na voljo povezavo s hitrostjo 1Gb/s, saj lahko v takih okoliščinah izmenjavo izvedemo v slabih 10-ih minutah.

10.1 Recikliranje protokola 1

Do sedaj smo obravnavali enkratno izvajane protokola 1. Pri večkratni ponovitvi protokola je čas dokazovanja in preverjanja vektorja W nezanemarljiv. Nastali problem rešimo tako, da RSA ključe generirano samo enkrat. Pri izmenjavi RSA ključev

	gen. RSA ključev	dokaz. prav. modula	prev. prav. modula
loopback	0.67 ± 0.04	0.25 ± 0.00	0.25 ± 0.00
1Gb/s		3.36 ± 0.05	3.31 ± 0.05
100Mb/s		3.77 ± 0.02	3.72 ± 0.02

Tabela 1: Potreben čas (v sekundah) za sestavljanje, preverjanje in dokazovanje pravilnosti konstrukcije javnega modula N. Vsaka stran po enkrat dokazuje pravilnost svojega javnega modula, po enkrat pa preverja veljavnost konstrukcije javnega modula soudeleženca.

	dokaz. prav. W	prev. prav. W	celotna izmen. ključ.
loopback	19.02 ± 0.08	20.19 ± 0.09	39.60 ± 0.13
1Gb/s	18.30 ± 0.06	19.45 ± 0.12	39.82 ± 0.10
100Mb/s	18.33 ± 0.10	19.49 ± 0.12	40.15 ± 0.11

Tabela 2: Potreben čas (v sekundah) za preverjanje in dokazovanje pravilnosti konstrukcije vektorja W ter za celotno izmenjavo ključev. Vsaka stran po enkrat dokazuje pravilnost svojega vektorja W, po enkrat pa preverja veljavnost konstrukcije vektorja W soudeleženca.

si morata Ana in Bojan dokazati (le enkrat), da javna modula sestavljata dve varni praštevili večji od 2^ℓ. V tem primeru potrebujemo pri dokazovanju in preverjanju vektorja W le en krog v protokolu 7. S tem se časovna zahtevnost zmanjša za približno 13-krat.

loopback	velikost sp.	gen. popač. vez.	eval. popač. vez.	čas celot. proto.
MD5	55B	2.64 ± 0.00	2.64 ± 0.00	46.60 ± 0.23
SHA-1		2.65 ± 0.01	2.65 ± 0.01	46.40 ± 0.26
SHA-256		2.66 ± 0.00	2.65 ± 0.00	46.60 ± 0.22
SHA3-256	55B	2.65 ± 0.01	2.65 ± 0.01	47.46 ± 0.50
	500B	2.79 ± 0.01	2.79 ± 0.01	46.27 ± 0.29
	5KB	4.01 ± 0.01	4.00 ± 0.01	48.20 ± 0.21
	50KB	16.08 ± 0.04	15.85 ± 0.04	72.19 ± 0.25
	500KB	158.48 ± 0.71	134.09 ± 0.70	357.03 ± 1.54

Tabela 3: Potreben čas (v sekundah) za izvajanje protokola "popačeno vezje" s strani generatorja in evalvatorja ter čas izvajanja celotnega protokola za različne specifikacije pri uporabi povratne zanke. Vsaka "stran" po enkrat sodeluje v protokolu "popačeno vezje" kot generator, po enkrat pa kot evalvator.

1Gb/s	velikost sp.	gen. popač. vez.	eval. popač. vez.	čas celot. proto.
MD5	55B	5.31 ± 0.01	5.30 ± 0.01	58.04 ± 0.24
SHA-1		5.30 ± 0.00	5.29 ± 0.01	57.61 ± 0.16
SHA-256		5.32 ± 0.01	5.32 ± 0.01	57.68 ± 0.33
SHA3-256	55B	5.31 ± 0.00	5.30 ± 0.01	57.93 ± 0.20
	500B	5.54 ± 0.01	5.51 ± 0.00	58.49 ± 0.92
	5KB	7.77 ± 0.05	7.74 ± 0.01	62.72 ± 0.14
	50KB	30.22 ± 0.20	29.98 ± 0.17	108.40 ± 0.10
	500KB	273.66 ± 2.77	251.32 ± 1.16	594.77 ± 1.30

Tabela 4: Potreben čas (v sekundah) za izvajanje protokola "popačeno vezje" s strani generatorja in evalvatorja ter za izvajanja celotnega protokola za različne specifikacije pri uporabi gigabitne povezave. Vsaka stran po enkrat sodeluje v protokolu "popačeno vezje" kot generator, po enkrat pa kot evalvator.

100Mb/s	velikost sp.	gen. popač. vez.	eval. popač. vez.	čas celot. proto.
MD5	55B	46.68 ± 0.40	46.66 ± 0.41	141.60 ± 0.35
SHA-1		46.45 ± 0.03	46.44 ± 0.03	141.76 ± 0.11
SHA-256		46.59 ± 0.04	46.58 ± 0.04	141.98 ± 0.66
SHA3-256	55B	46.65 ± 0.15	46.63 ± 0.14	142.31 ± 0.90
	500B	48.23 ± 0.08	48.19 ± 0.05	144.51 ± 0.24
	5KB	66.13 ± 0.12	66.12 ± 0.11	181.27 ± 0.26
	50KB	245.88 ± 1.33	245.64 ± 1.31	540.57 ± 0.45
	500KB	2064.75 ± 11.12	2042.42 ± 9.29	4178.04 ± 3.04

Tabela 5: Potreben čas (v sekundah) za izvajanje protokola "popačeno vezje" s strani generatorja in evalvatorja ter za izvajanja celotnega protokola za različne specifikacije pri uporabi 100-megabitne povezave. Vsaka stran po enkrat sodeluje v protokolu "popačeno vezje" kot generator, po enkrat pa kot evalvator.

	velikost sp.	prenos podatkov
MD5	55B	1.07GB
SHA-1		1.07GB
SHA-256		1.07GB
SHA3-256	55B	1.07GB
	500B	1.11GB
	5KB	1.50GB
	50KB	5.47GB
	500KB	45.18GB

Tabela 6: Količina prenesenih podatkov za različne specifikacije pri različnih velikosti sporočil. Prenos podatkov je vsota prejetih in poslanih.

11 Zaključek

Konstruirali smo protokol poštene postopne menjave sporočil s preverjanjem specifikacij brez posredovanja zaupanja vredne tretje osebe. Protokol temelji na shemah za časovne zaprisege in na Yaovem protokolu "popačeno vezje". Pokazali smo, da je tak protokol možno realizirati v realnem svetu za izmenjavo krajših sporočil (50KB oz. 500KB), če le imamo dovolj dobro mrežno povezavo med sodelujočima (100Mb/s oz. 1Gb/s). Pri izbiri zahtev (oz. specifikacij) smo se omejili na preverjanje zgostitve sporočila za bolj popularne kriptografske zgoščevalne funkcije (MD5, SHA-1, SHA-256 in SHA3-256).

V nadaljnjem razvoju aplikacije za izmenjavo sporočil bi bilo potrebno formalno dokazati varnost celoletnega protokola 1, izvesti analizo možnih časovnih napadov in uskladiti varnostne parametre, ker je trenutna izbira le-teh neučinkovita. Veliko se še da narediti na področju optimizacije univerzalnih gradnikov logičnega vezja C, ki pripadajo funkciji F. Celotno aplikacijo bi bilo treba implementirati v enem samem jeziku (predlagamo C++) tako, da bi jo bilo mogoče uporabiti na različnih operacijskih sistemih in platformah.

Ena izmed možnih aplikacij je popolnoma anonimna izmenjava digitalne lastnine v omrežju "vsak z vsakim" (angl. *peer to peer networking*, kratica P2P), o kateri imamo dovolj informacij, da tvorimo zahteve. Z uporabo tako implementirane aplikacije (ali z manjšimi modifikacijami) je dovolj imeti le zgostitev, ki pripada eni izmen omenjenih zgoščevalih funkcij za željeno sporočilo oz. digitalno lastnino. Pridobitev primerne zgostitve je nemogoča, če je sporočilo pošiljatelja unikatno. Za bolj znana sporočila oz. digitalne lastnine lahko pridobimo pripadajočo zgostitev prek javnih in zasebnih P2P omrežij za deljenje datotek ali pa jo prejmemo od neke ZVTO. Glavna prednost tega načina izmenjave je minimalna digitalna sled, saj za izvedbo izmenjave potrebujemo le varno povezavo med udeležencema. Slaba stran je, da je količina komunikacije vsaj linearno odvisna od kompleksnosti specifikacije in od same dolžine datoteke.

Zadnje čase je varno večstrankarsko računaje vedno bolj popularno, vedno več pa je tudi praktičnih realizacij. Zato menimo, da bo v prihodnosti mogoče pošteno izmenjati tudi daljša sporočila pri bolj kompleksnih specifikacijah.

Literatura

[1] G. Asharov in dr., *More efficient oblivious transfer and extensions for faster secure computation*, v: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM, 2013, str. 535–548.

[2] G. Avoine in dr., *Reducing fair exchange to atomic commit*, Technical report/Ecole Polytechnique Fédérale, Swiss Federal Institute of Technology (EPFL) **11** (2004).

[3] M. Bellare, V. T. Hoang in P. Rogaway, *Foundations of garbled circuits*, v: Proceedings of the 2012 ACM Conference on Computer and Communications Security, ACM, 2012, str. 784–796.

[4] D. J. Bernstein in dr., *High-speed high-security signatures*, Journal of Cryptographic Engineering **2**(2) (2012) 77–89.

[5] L. Blum, M. Blum in M. Shub, *A simple unpredictable pseudo-random number generator*, SIAM Journal on computing **15**(2) (1986) 364–383.

[6] M. Blum, *Coin flipping by telephone a protocol for solving impossible problems*, ACM SIGACT News **15**(1) (1983) 23–27.

[7] D. Boneh in M. Naor, *Timed commitments*, v: Advances in Cryptology — CRYPTO 2000, Springer Berlin Heidelberg, 2000, str. 236–254.

[8] D. Boneh in dr., *Twenty years of attacks on the RSA cryptosystem*, Notices of the AMS **46**(2) (1999) 203–213.

[9] L. T. A. N. Brandão, *Secure two-party computation with reusable bit-commitments, via a cut-and-choose with forge-and-lose technique*, v: Advances and Cryptology - ASIACRYPT 2013, Springer Berlin Heidelberg, 2013, str. 441–463.

[10] R. P. Brent in P. Zimmermann, *Modern computer arithmetic*, Cambridge University Press, 2010.

[11] N. Büscher in S. Katzenbeisser, *Compilation for Secure Multi-party Computation*, Springer, 2017.

[12] J. Camenisch in M. Michels, *Proving in zero-knowledge that a number is the product of two safe primes*, v: Advances in Cryptology – EUROCRYPT '99, Springer Berlin Heidelberg, 1999, str. 107–122.

[13] D. Chaum in T. P. Pedersen, *Wallet databases with observers*, v: Advances in Cryptology — CRYPTO' 92, Springer Berlin Heidelberg, 1993, str. 89–105.

[14] T. Chou in C. Orlandi, *The simplest protocol for oblivious transfer*, v: International Conference on Cryptology and Information Security in Latin America, Springer, 2015, str. 40–58.

[15] E. Clarke, D. Kroening in F. Lerda, *A tool for checking ANSI-C programs*, v: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2004, str. 168–176.

[16] D. Evans in dr., *A pragmatic introduction to secure multi-party computation*, Foundations and Trends® in Privacy and Security (2018) 70–246.

[17] K. A. Fathy, H. M. Bahig in A. A. Ragab, *A fast parallel modular exponentiation algorithm*, Arabian Journal for Science and Engineering (2018) 903–911.

[18] T. K. Frederiksen in dr., *Minilego: Efficient secure two-party computation from general assumptions*, v: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2013, str. 537–556.

[19] Y. Huang, J. Katz in D. Evans, *Efficient secure two-party computation using symmetric cut-and-choose*, v: Annual Cryptology Conference, Springer, 2013, str. 18–35.

[20] Y. Huang in dr., *Faster secure two-party computation using garbled circuits*, v: USENIX Security Symposium, **201**, 2011, str. 331–335.

[21] M. Jawurek, F. Kerschbaum in C. Orlandi, *Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently*, v: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, ACM, 2013, str. 955–966.

[22] M. Keller, E. Orsini in P. Scholl, *Actively secure OT extension with optimal overhead*, v: Annual Cryptology Conference, Springer, 2015, str. 724–741.

[23] V. Kolesnikov in T. Schneider, *Improved garbled circuit: Free XOR gates and applications*, v: International Colloquium on Automata, Languages, and Programming, Springer, 2008, str. 486–498.

[24] Y. Lindell, *Fast cut-and-choose-based protocols for malicious and covert adversaries*, Journal of Cryptology **29**(2) (2016) 456–490.

[25] Y. Lindell, *How to simulate it–a tutorial on the simulation proof technique*, v: Tutorials on the Foundations of Cryptography, Springer, 2017, str. 277–346.

[26] Y. Lindell in B. Pinkas, *Secure two-party computation via cut-and-choose oblivious transfer*, Journal of cryptology **25**(4) (2012) 680–722.

[27] H. Lipmaa, P. Rogaway in D. Wagner, *CTR-mode encryption*, v: First NIST Workshop on Modes of Operation, Citeseer, 2000.

[28] D. Micciancio in M. Walter, *On the bit security of cryptographic primitives*, v: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2018.

[29] P. L. Montgomery, *Modular multiplication without trial division*, Mathematics of computation **44**(170) (1985) 519–521.

[30] M. Naor, B. Pinkas in R. Sumner, *Privacy preserving auctions and mechanism design*, EC **99** (1999) 129–139.

[31] N. I. of Standards in Technology, *Digital signature standard (DSS)*, FIPS 186-4 (2013).

[32] N. I. of Standards in Technology, *Secure hash standard (SHS)*, FIPS 180-4 (2015).

[33] N. I. of Standards in Technology, *SHA-3 standard: Permutation-based hash and extendable-output functions*, FIPS 202 (2015).

[34] J. Poon in T. Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, 2016.

[35] M. O. Rabin, *How to exchange secrets with oblivious transfer*, IACR Cryptology ePrint Archive **2005** (2005) 187.

[36] P. Rogaway in S. Micali, *The round complexity of secure protocols*, doktorska disertacija, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1991.

[37] N. Smart, *Circuits of Basic Functions Suitable For MPC and FHE*, `https://homes.esat.kuleuven.be/~nsmart/MPC/old-circuits.html`, 2017.

[38] N. Smart in dr., *"Bristol Fashion" MPC Circuits*, `https://homes.esat.kuleuven.be/~nsmart/MPC/`, 2017.

[39] D. Stinson in M. Paterson, *Cryptography: Theory and Practice*, CRC Press, 2018.

[40] X. Wang, A. J. Malozemoff in J. Katz, *EMP-toolkit: Efficient MultiParty computation toolkit*, `https://github.com/emp-toolkit`, 2016.

[41] G. Wood in dr., *Ethereum: A secure decentralised generalised transaction ledger*, Ethereum project yellow paper **151**(2014) (2014) 1–32.

[42] A. C.-C. Yao, *How to generate and exchange secrets*, v: 27th Annual Symposium on Foundations of Computer Science, IEEE, 1986, str. 162–167.

[43] S. Zahur, M. Rosulek in D. Evans, *Two halves make a whole*, v: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2015, str. 220–250.

[44] *PyCryptodome*, `https://github.com/Legrandin/pycryptodome`, 2016.