Pobjednici natječaja CAESAR
Tehnička dokumentacija
Verzija <1.3>

Studentski tim: Ivana Dasović
Marta Knežević
Vinko Sabolčec
Karlo Šutalo

Nastavnik: dr. sc. Marin Golub

Sadržaj

1. UVOD

1.1 CAESAR natječaj

CAESAR natječaj je natječaj u razvoju novih algoritama autentifikacijske kriptografije. Algoritmi se dijele u 3 kategorije: algoritmi za uređaje s ograničenim resursima, algoritmi za primjenu u uređajima s visokim performansama i algoritmi s dubinskom obranom koji u više slojeva imaju sigurnosne kontrole. Natječaj se temelji na kriptografiji s tajnim ključem. Preteća CAESAR natječaju su otvoreno natjecanje za novi Napredni Enkripcijski Standard, eSTREAM- ECRYPT Stream Chiper Project koji je zahtjevao podnošenje novih stream kriptiranih poruka prikladnih za široku uporabu te otvoreno natjecanje za novi hash standard SHA-3. Kriptografija tajnim ključem štiti povjerljivost i integritet poruka od bilo kakvog lošeg ponašanja posrednika. Većina kriptiranih i povjerljivih podataka zaštićena je ili hibridno, javnim ključem i tajnim ključem, ili samo tajnim ključem. SUPERCOP je alat za mjerenje performansi kriptografskih software- a poput hash funkcija, sustava enkripcije javnim ključem, sustava za potpisivanje javnim ključem. Specijalno SUPERCOP uzima u obzir kriterije poput vremena potrebnog za kriptiranje kratkog paketa, vremena potrebnog za kriptiranje paketa srednje veličine, vremena potrebnog za kriptiranje dugačke poruke, duljine tajnog ključa i duljine javnog ključa. Natječaj ima nekoliko krugova te algoritmi koji nisu uspjeli zadovoljiti kriterije za određeni krug ne prolaze dalje. Obzirom na to da CAESAR natječaj nije projekt standardizacije algoritmi koji prođu u završni krug ne smatraju se standardima.

1.2 Rezultati projekta

1.2.1 Program

Glavni rezultat ovog projekta je program koji šifrira ili dešifrira unesene poruke ili datoteke, ovisno o načinu rada koji se odabere, te ispisuje izlaz ili grešku ukoliko je do nje došlo. Algoritmi koji su implementirani u program su:

- ACORN – 128
- ASCON
- Deoxys – II
- KETJE
- Aegis

Član tima koji je bio zadužen za programsko rješenje je Vinko Sabolčec.

1.2.2 Web stranica

Web stranica projekta napravljena je u edukativne svrhe te prikazuje kratak opis svih algoritama i omogućuje preuzimanje gotovog programskog rješenja, prezentacije i tehničke dokumentacije. Članica tima zadužena za izradu web stranice je Marta Knežević.

1.2.3 Prezentacija

U svrhu prezentiranja projekta napravljena je i PowerPoint prezentacija koja ukratko opisuje natječaj CAESAR, zadane algoritme i programsko rješenje. Članica tima zadužena za prezentaciju je Ivana Dasović.

2. ANALIZA ALGORITAMA

2.1 ACORN- 128

Ulazni podaci za funkciju enkripcije ACORN-128 su:

- Jasan tekst P duljine do 2^{64} bita
- Tajni ključ K duljine 128 bita
- Pridruženi podaci AD duljine do 2^{64} bita
- Javni broj poruke N (nonce) duljine 128 bita

Izlazni podaci funkcije za enkripciju ACORN-128 su:

- Šifrirana poruka C duljine |P| bita
- Autentifikacijska oznaka T duljine 128 bita

ACORN-128 kao IV koristi javni broj poruke N (nonce).
Stanje algoritma u i-tom koraku S_i ACORN-128 je opisano sa 293 bita.

Prilikom rada, ACORN koristi vektor bitova f gdje je sa f_i opisan bit i-tog koraka algoritma, vektor *keystream* bitova ks gdje je sa ks_i opisan bit i-tog koraka algoritma te vektore kontrolnih bitova ca i cb gdje su sa ca_i i cb_i opisani kontrolni bitovi i-tog koraka algoritma.

Funkcija za generiranje novog stanja izračunava novo stanje S_{i+1} na temelju trenutnog stanja S_i, podatkovnog bita m_i te kontrolnih bitova ca_i i cb_i. Funkcija na početku modificira stanje S_i koristeći šest *LFSR*-a (*linear-feedback shift register*) te koristi to modificirano stanje za generiranje *keystream* bita i *feedback* bita te nakon toga posmiče registar stanja za jedan bit te zadnji bit izračunava pomoću *feedback* bita f_i i podatkovnog bita m_i.

U svakom koraku se provodi generiranje *keystream* bita za i-ti korak ks_i pomoću modificiranog stanja algoritma u i-tom koraku S_i.

U svakom koraku se provodi i generiranje *feedback* bita za i-ti korak f_i pomoću modificiranog stanja u i-tom korkau S_i te kontrolnih bitova ca_i i cb_i.

Algoritam započinje inicijalizacijom. Inicijalizacija učitava ključ K i inicijalizacijski vektor IV (jedan par ključa K i inicijalizacijskog vektora IV se koriste za sigurno šifriranje poruke) u stanje i vrti se 1792 koraka šifriranja. U inicijalizaciji su kontrolni bitovi cb_i postavljeni u 1 pa se za generiranje novog stanja koristi *keystream* bit.

Sljedeći korak je procesiranje pridruženih podataka AD. *AD* se učitava u m (od nultog bita do |*AD*| – 1 bita). Iza toga se u m stavlja uzorak od 256 bita koji započinje jedinicom, a svi ostali bitovi su nule (1000...0000). Kontrolni bitovi u cb_i su postavljeni u 1 da se za generiranje novog stanja koristi ks_i. Kontrolni bitovi ca_i su postavljeni tako da se odvoji *AD* od poruke koji se šifrira ili šifrirane poruke, odnosno da se onemogući korištenje dijela *AD* kao poruke ili obrnuto.

Nakon toga slijedi šifriranje. U podatkovne bitove m se upisuje nešifrirana poruka te iza nje uzorak od 256 bita koji započinje jedinicom, a svi ostali bitovi su nule (1000...0000). U svakom koraku se računa novo stanje te se j-ti bit šifrirane poruke računa kao operacija eksluzivni ili između j-tog bita nešifrirane poruke i *keystream* bita za i-ti korak algoritma. Bitovi cb_i su postavljeni u 0 tako da se ne koristi *keystream* bit ne koristi za generiranje novog stanja. Kontrolni bitovi ca_i su postavljeni tako da se odvoji procesuiranje nešifiranog poruke ili šifrirane poruke kao autentifikacijske oznake u sljedećem koraku.

Nakon enkripcije se generira autentifikacijska oznaka T. U podatkovne bitove m_i se stavlja 768 bitova nula. 768 puta se računa novo stanje. Kontrolni bitovi cb_i su postavljeni u 1 tako da se koristi *keystream* bit za generiranje novog stanja. Autentifikacijska oznaka T se računa kao zadnjih |T| bitova (|T| <= 128) *keystream* ks bitova.

Dekripcija i verifikacija ima sličan postupak s napomenom da ukoliko verifikacija ne prolazi uspješno, algoritam ne smije vratiti autentifikacijsku oznaku i dešifriranu poruku.

ACORN je bitovno orijentiran sekvencijski autentifikacijski algoritam. Ne koristi blokove bitova te zbog toga nije potrebno prenositi duljinu parametara i dodatavati punjenje. Za sigurno korištenje algoritma, ključ treba biti

generiran na siguran i nasumičan način te se jedna kombinacija ključa i inicijalizacijskog vektora smiju koristiti za šifriranje samo jedne poruke. Također, jedan par ključa i inicijalizacijskog vektora se koriste s jednom fiksnom duljinom autentifikacijske oznake.
Inicijalizacija ACORN-a je dizajnirana za sprječavanje *linear* napada, *differential* napada i *cube* napada. S obzirom na to da ACORN nije blokovno orijentiran(?), napadi karakteristčni za algoritme koji jesu ne mogu direktno biti korišteni na ACORN. Stanje se ažurira na ne-linearan način te je zbog toga ACORN otporan na klasične napade na šifriranje.

ACORN je otporan na *guess-and-determine* napade te *forgery* napade.

ACORN je dizajniran kao *lightweight* algoritam. U jednom koraku se procesuira točno jedan bit te se zbog toga lako izvodi jednostavna hardverska implementacija. ACORN dozvoljava paralelizaciju 32 koraka istovremeno što znatno ubrzava izvođenje na hardverskim i softverskim implementacijama. Softverska implementacija ACORN-a ima mali broj linija koda te zbog toga pogoduje sustavima s ograničenom količinom memorije.

2.2 AEGIS

Preporučene vrijednosti parametara za AEGIS algoritme:

	AEGIS-128	AEGIS-128L	AEGIS-256
Ključ/[bit]	128	128	256
Inicijalizacijski vektor/[bit]	128	128	256
Stanje/[bit]	640	1024	768
Oznaka/[bit]	128	128	128
Maksimalna duljina poruke/[bit]	2^{64}	2^{64}	2^{64}

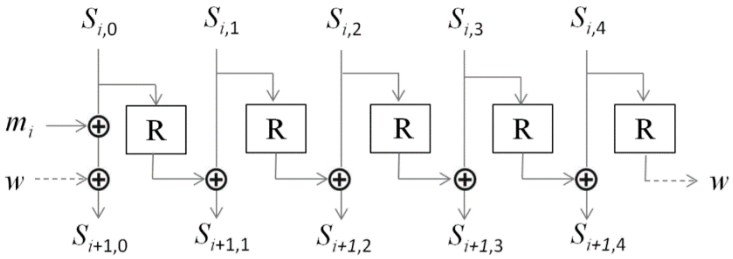
Opis varijabli i konstanti:

const	Konstanta
AD	Pridruženi podatci
C	Kriptirani tekst
IV	Inicijalizacijski vektor
K	Ključ
P	Običan tekst
S_i	Stanje na početku i-tog koraka
$S_{i,j}$	j-tih 16 bajtova stanja S_i
T	Autentifikacijska oznaka

AEGIS algoritmi koriste AESRound funkciju koja se sastoji od 3 manje funkcije: SubBytes, ShiftRows, MixColumns. SubBytes prima na ulazu 128 bitova odnosno 16 bajtova te svaki od njih mijenja s određenom vrijednosti iz S-tablice koja je konstantna. Izlaz SubBytes funkcije je 4x4 matrica koja je ujedno i ulaz u ShiftRows funkciju. Redci se pomiču u lijevo tako da se prvi redak ne pomiče, drugi redak se pomiče za jednu poziciju (bajt), treći redak za dvije pozicije i četvrti redak za tri pozicije. Vrijednosti koje „ispadnu“ ponovno se vraćaju u matricu s desne strane. Izlaz ShiftRows funkcije je 4x4 matrica koja ima iste elemente kao matrica s ulaza, ali su u drugačijem međusobnom odnosu. Funkcija MixColumns vrši transformaciju nad stupcima matrice. Kao ulaz funkcija dobiva 4 bajta iz jednog stupca te primjenom specijalne matematičke funkcije generira 4 potpuno nova bajta koji mijenjaju ulazne bajtove. Izlaz funkcije je 4x4 matrica koja se sastoji od potpuno novih elemenata.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Slika 2 Rijndael S-box



Slika 1 Ažuriranje stanja

Izvođenje AEGIS algoritama odvija se u koracima. Funkcija koja se izvodi pri svakom koraku je StateUpdate koja služi za ažuriranje stanja određenog koraka algoritma.

Za algoritam AEGIS-128 funkcija StateUpdate128() koja se koristi nad 80-bajtnim stanjem i 16-bajtnim blokovima poruke, a definirana je kao $S_{i+1} = StateUpdate128(S_i, m_i)$.

Za algoritam AEGIS-256 koristi se funkcija StateUpdate256() koja radi na istom principu kao i StateUpdate128, ali djeluje nad 96-bajtnim stanjem pa je podijeljena na 6 dijelova. Definirana je kao $S_{i+1} = StateUpdate256(S_i, m_i)$.

Za algoritam AEGIS-128L koristi se funkcija StateUpdate128L() koja djeluje nad 128-bajtnim stanjem i dvjema 16-bajtnim blokovima poruka. Definirana je kao $S_{i+1} = StateUpdate128L(S_i, m_a, m_b)$.

Rad AEGIS algoritama podijeljen je u 5 faza, a one su: inicijalizacija podataka, procesiranje pridruženih podataka, kriptiranje, generiranje autentifikacijske oznake i dekriptiranje. Inicijalizacija podataka služi kako bi se inicijalizacijski vektor i ključ učitali u stanje. Sastoji se od deset iteracija ažuriranja stanja u kojima se inicijalizacijski vektor koristi kao poruka.

Prvi korak je inicijalizacija podataka koja služi učitavanju inicijalizacijskog vektora, ključa i konstanti u stanje. Drugi korak služi za određivanje poruke koja se koristi u ažuriranju stanja u i-toj iteraciji inicijalizacije. U trećem koraku se u iteracijama ažurira stanje. Broj iteracija inicijalizacije podataka ovisi o odabiru algoritma iz razloga što algoritmi imaju drugačiju duljinu stanja

Faza procesiranja pridruženih podataka služi kako bi se u stanje dodali pridruženi podatci koji služe kao kontekst poruci koju kriptiramo. Potrebno je provjeriti je li zadnji blok pridruženih podataka puni blok, odnosno ima li 128 bitova za AEGIS-128 i AEGIS-256, odnosno 256 bitova za AEGIS-128L te ako nema blok se dopunjuje nulama. Ako je duljina pridruženih podataka (*adlen*) nula, stanje se neće ažurirati.

U fazi kriptiranja svaki se blok poruke koristi za ažuriranje stanja i za dobivanje kriptiranog bloka. Potrebno je provjeriti je li zadnji blok poruke puni blok, odnosno ima li 128 bitova za AEGIS-128 i AEGIS-256, odnosno 256 bitova za AEGIS-128L te ako nema blok se dopunjuje nulama.

Nakon faze kriptiranja potrebno je generirati autentifikacijsku oznaku koja će osigurati nepromijenjenost podataka. Generiranje oznake događa se u 3 koraka i koriste se duljine pridruženih podataka i poruke. U prvom koraku definiramo varijablu *temp* koja će nam služiti kao poruka u ažuriranju stanja pri generiranju oznake, a ta varijabla se sastoji od bloka stanja XOR *adlen* || *msglen*, gdje su *adlen* i *msglen* reprezentirani kao 64-bitni cijeli brojevi. U drugom koraku ažuriramo stanje pomoćnom varijablom *temp*. Zadnji korak je generiranje oznake iz blokova zadnjeg stanja operacijom XOR. Autentifikacijska oznaka T je prvih t bitova oznake T'.

Da bi dekriptiranje bilo moguće moraju biti poznati podatci o duljini ključa, duljinu inicijalizacijskog vektora, i duljinu oznake. Početak dekriptiranja sastoji se od inicijalizacije podataka i procesiranja pridruženih podataka. Nakon toga potrebno je provjeriti je li zadnji blok kriptiranog teksta puni blok te ako nije popunjava se nulama. U idućem koraku se događa dekriptiranje pojedinih blokova kriptiranog teksta koji se koriste i za ažuriranje stanja. Provjera autentifikacijske oznake radi se na način da se iz dekriptiranog teksta generira nova oznaka te se provjeri jednakost dobivenih oznaka.

Kako bi se dobila tražena svojstva algoritama za kriptiranje, postoje zahtjevi koji moraju biti zadovoljeni. Prvi zahtjev algoritama jest da se ključ mora generirati slučajno iz uniformnog prostora ključeva. Također, poruke koje se kriptiraju parom (ključ, IV) moraju imati istu duljinu autentifikacijske oznake i moraju imati jedinstveni par za svaku poruku. Zadnje svojstvo koje mora biti zadovoljeno jest da, ako kriptirani tekst ne prođe verifikaciju, dekriptirani tekst i kriva autentifikacijska oznaka ne smiju biti vidljivi na izlazu. Korištenje AEGIS algoritama smatra se sigurnim ako su zadovoljena sva tri uvjeta. U sigurnim uvjetima korištenja, prednost napadača u *forgery attack*-u je 2^{-t}, a ako se napad ponovi n puta onda je $n \times 2^{-t}$. Ako napadač nije uspio otkriti informacije tijekom *forgery attack*-a, nije moguće razotkriti ključ i stanje brže od iscrpnog pretraživanja ključeva. Preporučeno je korištenje autentifikacijske oznake duljine 128 bitova, no bitno je uočiti da prilikom korištenja AEGIS-256 algoritma s oznakom duljine 128 bitova postoji način pronalaska stanja brže od iscrpnog pretraživanja ključeva ako se napravi *forgery attack* oko 2^{128} puta za isti par (ključ, IV).

Cilj ovih algoritama jest visoka sigurnost i visoke performanse kriptiranja. Visoke performanse postignute su korištenjem *AESRound* funkcije koja je implementirana u najnovijim Intel i AMD procesorima. Uz to, algoritam je dizajniran tako da koristi što više paralelnih *AESRound* poziva kako bi se ubrzao rad algoritma. Visoka sigurnost algoritama postiže se stvaranjem nasumičnih razlika u inicijalizacijskom vektoru i autentifikacijskoj oznaci u koraku inicijalizacije stanja kako *„differential attack“* ne bi bio moguć.

2.3 ASCON- 128

Algoritam ASCON- 128 pripada familiji $\mathrm{ASCON}_{a,b}$-k-r algoritama za autentificiranu enkripciju.
Ulazni podaci za funkciju enkripcije su:

- Jasan tekst P - podijeljen u blokove podataka P_i duljine *r* bitova
- Asocirani podaci A - podijeljeni u blokove podataka A_i duljine *r* bitova
- Tajni ključ K s *k* bitova
- Javni broj poruke (nonce) N s *k* bitova. (preporučena konfiguracija 128 bitova)

Pri procesu enkripcije koristi se inicijalizacijski vektor IV koji je specifičan i unaprijed određen za algoritam. IV za Ascon-128 je 80400c0600000000, a računa se pomoću duljina ključa, duljine r (rate) i broja rundi *a* i *b* permutacija. Pri procesu enkripcije ne koristi se tajni broj poruke tj. njegova duljina je 0.
Funkcija enkripcije na svom izlazu daje:

- šifrirani tekst C koji je iste duljine kao početni tekst P - podijeljen u blokove podataka C_i duljine *r* bitova
- autentificirajuću oznaku T s *k* bitova. (preporučena konfiguracija 128 bitova)

Parametri kod enkripcije koristeći algoritam Ascon- 128 su :

- Duljina ključa $k \leq 128$ bitova (preporučena konfiguracija 128 bitova)
- Rate *r* (broj bitova koji označava duljine blokova) (preporučena konfiguracija 64 bitova)
- Brojevi permutacija *a* i *b* (preporučene konfiguracije redom 12 i 6 bitova)

Pri procesu dekriptiranja koriste se ključ K, nonce N, asocirani podaci A, šifrirani tekst C i autentificirajuća oznaka T. U slučaju da je oznaka ispravna, funkcija dekriptiranja vraća tekst P.

Ascon- 128 algoritam temelji se na konstrukciji sličnoj MonkeyDuplex, no koristi snažnije funkcije inicijalizacije

i finalizacije. Procesi enkripcije i dekripcije sastoje se od po četiri faze. Prva faza je inicijalizacija. Početno stanje S od 320 bita formira se pomoću konkateniranja inicijalizacijskog vektora (IV), tajnog ključa i javnog broja poruke. Stanje S sastoji se od 5 registara s po 64 bita. Inicijalno stanje permutira se *a* puta, te se na kraju napravi operacija *xor* s ključem K.

Nakon inicijalizacije slijedi faza procesiranja asociranih podataka. U toj fazi podatci koji su nastali operacijom *xor* između prvih r bitova stanja S i bloka podataka A_i, te konkatenacijom ostatka stanja S permutiraju se b puta. Na kraju svake permutacije postupak se nastavlja s novim blokom A_i sve dok ima novih blokova. Na kraju faze napravi se *xor* operacija stanja S s brojem 1.

Sljedeća faza je faza enkripcije/dekripcije. Slično kao i u prethodnoj fazi podaci koji su nastali operacijom *xor* između prvih r bitova stanja S i bloka podataka P_i, te konkatenacijom ostatka stanja S permutiraju se b puta. Kod enkripcije blok podataka šifriranog teksta (C_i) određen je operacijom *xor* između prvih r bitova stanja S i bloka podataka P_i. Posljednji blok podataka nastao operacijom *xor* se ne permutira b puta. Kod dekriptiranja, operacija *xor* obavlja se između šifriranog teksta i stanja, a rezultat operacije je blok početnog teksta P_i.

Posljednja faza je faza finalizacije. Na početku faze napravi se operacija *xor* između stanja S i konkatenacije ključa i znamenki 0. Takvo stanje prolazi kroz posljednji proces permutacije koji se ponavlja *a* puta. *Xor* između izlaza stanja S i ključa K je autentificirajuća oznaka T.

Proces permutiranja sastoji se od 3 dijela: dodavanja konstanti, operacijama nad vertikalnim podacima stanja (substitution layer) i operacijama nad horizontalnim podacima stanja (linear diffusion layer).

Kod dodavanja konstanti unaprijed određene konstante dodaju se (operacija *xor*) u srednji registar x2 stanja S. U substitution layer-u izvode se operacije nad dijelovima stanja S određenim registrima tj. nad različitim podacima duljine 64 bita (*xor, and, not* nad registrima međusobno). U linear diffusion layer-u izvode se operacije unutar pojedinog podatka duljine 64 bita (*xor* i rotacija bitova).

Unutarnje permutacije u Ascon-u temelje se na jednostavnim st*and*ardnim operacijama na 64-bitnim podacima. Te operacije izvedive su i na procesorima s malom veličinom riječi. Substitution layer i linear layer dizajnirani su tako da podržavaju paralelno izvođenje operacija. Do pet instrukcija može se paralelno izvoditi u svakoj fazi permutacije.

Šifriranje se može odvijati bez znanja o duljini teksta. Dešifriranje se također može odvijati na blokovima podataka zasebno, te nije potrebno saznanje o duljini šifriranog teksta.

Enkripcija i dekripcija odvijaju se s jednim prolazom kroz podatke.

Pri implementaciji Ascon-a nisu potrebne inverzne operacije tj. permutacije se odvijaju uvijek u jednom smjeru, što smanjuje zahtjevnost implementacije.

S obzirom na to da sadrži broj poruke (nonce) ponavljanje tog broja dovodi do ugrožavanja sigurnosti algoritma jer je moguće detektirati slične prefikse teksta. Prema tome uvjet korištenja Ascona je jednostruko korištenje javnog broja poruke.

S obzirom na malu veličinu stanja S i jednostavnu strukturu funkcija, Ascon ima vrlo dobre karakteristike po pitanju veličine i brzine, te se stoga može efikasno koristiti u hardware implementacijama koje zahtijevaju malo memorije ili veliku brzinu.

Ascon omogućava jednostavnu implementaciju na hardware-u i software-u uz dobre performanse. Idealan je za situacije gdje mnogo uređaja komunicira s backend serverom kao u Internet of Things.

Ascon je jedan od najbržih algoritama kriptiranja s natječaja CAESAR za kratke poruke. Inicijalizacija i finalizacija su puno jednostavnije u usporedbi s većinom algoritama koji šifriraju blokove podataka.

Mala veličina stanja S omogućava pohranu cijelog stanja u registrima procesora na većini platformi, čime reducira dohvaćanje podataka iz cache-a i memorije. Ta funkcionalnost može biti korisna kod cloud aplikacija kako bi se spriječili napadi na cache poput cross-VM napada.

2.4 Deoxys- II- 128

Ulazni podaci za funkciju enkripcije Deoxys su:

- Jasan tekst M proizvoljne duljine
- Tajni ključ K duljine 128 ili 256 bita (128 bita za Deoxys-II-128)
- Pridruženi podaci AD duljine proizvoljne duljine (128 bita za Deoxys-II-128)
- Javni broj poruke N (nonce) duljine 120 bita

Izlazni podatci funkcije za enkripciju Deoxys su:

- Šifrirana poruka C duljine |M| bita
- Autentifikacijska oznaka T duljine |T| bita (preporuka |T| = 128 bita)

Šifriranje Deoxys-II (način dozvole ponavljanja broja poruke (*nonce*)) započinje inicijalizacijom *associated data* AD tako da se podijeli u blokove od 128 bita. Interni vektor bitova *Auth* se inicijalizira na 0. Za svaki blok AD_i se računa *Auth* kao *Auth* = *Auth* XOR E_K(enkodiran broj bloka *i*, blok AD_i). E_K je funkcija *tweakable block cipher*-a Deoxys BC. Nakon toga se na sličan način podijeli ulazna poruka M u blokove od 128 bita. U autentifikacijsku oznaku T se stavlja izračunati *Auth* te se za svaki blok poruke M_i računa novi T = T XOR E_K(enkodiran broj bloka *i*, blok M_i). Na kraju slijedi generiranje šifirane poruke. Za svaki blok poruke M_i se računa blok šifrirane poruke C_i kao C_i=M_i XOR E_K(enkodirana oznaka, javna poruka N). Rezultat se vraća kao konkatenacija blokova šifrirane poruke C_i i autentifikacijske oznake T.

Dešifriranje Deoxys-II algoritmom se provodi slično kao i šifriranje. Na početku se šifrirana poruka C i *associated data* AD podijele u blokove od 128 bita. Poruka se dešifrira analogno šifriranju. Na isti način kao i kod šifriranja se izračunava interni vektor *Auth* pomoću kojeg se ponovno izračunava autentifikacijska oznaka T′. Ako su dobivena autentifikacijska oznaka T i izračunata autentifikacijska oznaka T′ jednaki, vraća se dešifrirana poruka, a u suprotnom se ne vraća ništa.

Deoxys BC je *tweakable block ciper* koji je dizajnom sličan AES-u. Kao parametre prima poruku P, ključ K i parametar *tweak* T. Ima 128-bitno stanje i proizvoljne duljine ključa i *tweaka*. Za funkciju E_K Deoxys BC-a vrijedi $E_K(T, P) = C$ i $E_K^{-1}(T, C) = P$.

Deoxys ima vrlo dobre sigurnosne karakteristike. Sigurnost je mjerena u broju rundi (faza) kroz koje podaci prolaze dok nisu potpuno sigurni. Deoxys-BC-256 zahtijeva 14, a Deoxys-BC-384 16 rundi, dok je za najbolje napade na algoritme s AES dizajnom koji imaju sličnu veličinu ključa potrebno 7 do 9 rundi. S 14 i više rundi zaštite podataka, Deoxys nudi pouzdanu sigurnost bez obzira na vrstu napada.

Deoxys je efikasan kod kriptiranja malih podataka, što je važno kod mnogo jednostavnih aplikacija gdje je veličina poruka pretežito mala. Za razliku od većine algoritama koji se koriste za šifriranje kratkih poruka, Deoxys nema zahtjevnu implementaciju u fazi inicijalizacije. Njegova efikasnost temelji se na modificirajućem blokovskom šifriranju koje izbjegava bilo kakvo inicijalno računanje.

Deoxys karakterizira jednostavna i razumljiva konstrukcija i implementacija. Dodatna karakteristika algoritma je fleksibilnost koja omogućuje korisniku proizvoljan odabir veličina ključa i modificirajućih podataka.

Teoretske performanse algoritma su optimalne. Deoxys-II zahtjeva 2m + 1 poziva internog šifriranja za m blokova od po n bitova.

Deoxys je otporan na *side-channel* napade, a pri tome koristi iste tehnike kao AES algoritmi.

Ponavljanje broja poruke (*nonce*) može ugroziti efikasnost algoritma, no samo u slučajevima velikog ponavljanja, što dozvoljava korištenje istog broja poruke (*nonce*) nekoliko puta.

Algoritam Deoxys može se koristiti u Internetskom prometu kao i kod lightweight aplikacija gdje su veličine paketa koje se šalju dosta male.

Deoxys pokazuje dobre performanse kod implementacije u software-u. Kao i većina ostalih algoritama temeljenih na AES dizajnu, vrlo se uspješno izvodi na novijim procesorima koji imaju implementiran novi AES-NI set instrukcija. Dodatno, uz korištenje paralelizma broj ciklusa po bajtu podataka pri kriptiranju se uvelike smanjuje.

2.5 KETJE

	K (ključ) [bytes]	Nonce [bytes]
KETJE SR	do 382	382 - |K|
KETJE JR	do 182	182 - |K|

Varijabilna duljina poruke

KETJE algoritmi koriste niz operacija nazvan KECCAK- p permutacije koje su nastale iz KECCAK- f funkcije, a izvode se nad stanjem A. Širina permutacije iznosi b = 25 * 2 ^l, broj rundi je n_r te se stanje A tada prikazuje trodimenzionalnom matricom [5, 5, w] gdje je w = 2 ^l. Svi blokovi A duljine su p bitova, osim posljednjeg koji može imati i manje bitova, ali ne smije biti prazan. Nadalje, KETJE algoritmi oslanjaju se na MonkeyDuplex konstrukciju koja održava stanje i ima broj rundi n_r . Ulaz i izlaz su string s tim da izlazni string ovisi o svim dosadašnjim ulazima. Ulazni niz duljine l proširuje se multi- rate paddingom do duljine b. Tijekom pokretanja MonkeyDuplex sadrži ključ i nonce. Autentifikaciju poruka osigurava MonkeyWrap koji se zasniva na MonkeyDuplex konstrukciji te kao ulaz prima zaglavlje A i tijelo poruke B i vraća kriptiranu poruku C i oznaku T. MonkeyWrap podržava sesije, tj. enkripciju niza poruka kod koje je oznaka T za svaku poruku autentična obzirom na cijeli niz poruka, nonce jedinstven za cijelu sesiju te su poruke procesuirane redom zbog verifikacije oznaka T. Nakon inicijalizacije MonkeyWrap u okvir dodaje 2 bita za svaki ulazni blok kako bi se osiguralo razdvajanje domena. Key pack se sastoji od prvog bajta koji prikazuje duljinu cijelog key packa u bajtovima, ključa čija je duljina ograničena na višekratnike broja 8 i jednostavnog padding. Nonce sadrži proizvoljan broj te se tijekom enkripcije koristi samo jednom.

KETJE algoritam je set od 4 autentične funkcije za enkripciju: KETJE SR, KETJE JR, KETJE MINOR te KETJE MAJOR. KETJE SR koristi ključ K varijabilne duljine do 382 bita i nonce duljine do 382 - |K| bitova, KETJE JR koristi ključ K duljine do 182 bita i nonce duljine do 182 - |K| bitova. KETJE MINOR I KETJE MAJOR koriste izvrnutu permutaciju KECCAK- p* kako bi odradile više linija po rundi od KETJE JR i KETJE SR. KETJE MINOR i KETJE MAJOR koriste ključ K duljine b – 18 bitova i nonce duljine b - |K| - 18 bitova. Navedene funkcije većinom koriste samo jednu rundu KECCAK- p funkcije. Prilikom korištenja ovih algoritama zahtjeva se jedinstvenog ključa i noncea. KETJE omogućuje autentičnu enkripciju poruke M i asociranih podataka AD koristeći javni broj poruke N i ključ K. Stvara se KETJE objekt W s ključem K i javnim brojem poruke N te se zatim koristeći MONKEY WRAP zamotaju asocirani podaci i poruka pri čemu se dodaju po 2 bita za svaki ulaz čime se osigurava separacija domene i dobije se oznaka T s duljinom jednakom ciljanoj snazi sigurnosti s.

Osnovna vodilja četiriju KETJE prijedloga je maksimizacija kapaciteta nadoknađivanjem gubitka performanski smanjenjem rundi u KECCAK- p funkciji. Obzirom na to da KETJE koristi MonkeyDuplex umjesto običnog duplexa, MonkeyWrap i KECCAK- p funkcije, zaštićen je od side channel napada i u hardware- u i u software- u. Vjerojatnost uspjeha pronalaženja stanja A pogađanjem je minimalna stoga prilikom izvođenja svih operacija napadač nema pristup unutarnjem stanju. KETJE pruža autentičnu enkripciju, zahtjeva relativno malen broj računskih operacija, nudi kompetente performanse hardware- a i podupire sesije čime povećava područje primjene.

KETJE se primjenjuje u uređajima s malim memorijskim resursima. Između ostalog, služi za sigurno slanje poruka koristeći sigurne čipove poput pametnih kartica. Podržavanje sesija omogućuje jednostavan i agilan način slanja niza naredbi u obliku skripte sprječavajući napadača da ubaci, makne ili zamijeni naredbe u skripti. Algoritam se može implementirati u male hardware- a i mali kod za 8- bitne procesore pri čemu je velika brzina izvođenja.

Algoritam implementira funkcije koje se mogu više puta pozvati nad istim podacima i koje mogu biti ponovno iskorištene za neku drugu simetričnu enkripciju.

2.6 KEYAK

Broj rundi	n_r [bytes]	12
Kapacitet	c[bytes]	256
Veličina oznake T	τ[bytes]	128

	W (jedinica poravnavanja) [bytes]	l (key pack) [bytes]
RIVER KEYAK	32	36
LAKE KEYAK	64	40
SEA KEYAK	64	40
OCEAN KEYAK	64	40
LUNAR KEYAK	64	40

Veličina poruke i nonce (nuncij) varijabilne duljine

KEYAK je familija algoritama koja se zasniva na Motorist autentificirajućem enkripcijskom modulu i KECCAK- p permutacijskim funkcijama. KECCAK- p permutacijske funkcije predstavljaju niz operacija koje su nastale iz KECCAK- f funkcija, a izvode se nad stanjem A. Širina permutacije iznosi b = 25 * 2 ^l, broj rundi je n_r te se stanje A tada prikazuje trodimenzionalnom matricom [5, 5, w] gdje je w = 2 ^l. Svi blokovi A duljine su p bitova, osim posljednjeg koji može imati i manje bitova, ali ne smije biti prazan. Motorist modul ima spužvastu konstrukciju i troslojnu arhitekturu. Svaki sloj Motorist modula zadužen je za osiguravanje različitih funkcionalnosti te kao ulaz prima niz bajtova. Piston je entitet najnižeg sloja, koristi funkciju za permutacije te tijekom operacija nad podacima ne radi razliku između ulančanih vrijednosti i meta podataka. Koristi offset s 4 fragmenta: EOM je fragment koji označava broj bajtova u sljedećem izlaznom bloku i odvaja poruke, Crypt End fragment označava kraj fragmenta poruke u trenutnom ulaznom bloku, Inject Start fragment označava početak meta podataka u trenutnom ulaznom bloku (jednak je nuli ukoliko trenutni ulazni blok sadrži samo meta podatke) te Inject End kodira kraj meta podataka fragmenta u trenutnom ulaznom bloku. Funkcija Piston.Crypt omogućuje enkripciju ili dekripciju, ovisno o oznaci decrypt koja označava koja se operacija treba izvest. Funkcija Piston.Inject ubacuje meta podatke iz ulaznog niza kodirajući pritom početak meta podataka s offsetom Inject Start i kraj meta podataka s offsetom Inject End te u konačnici ponovno postavi offsete u njihovo početno stanje. Funkcija Piston.Spark primjenjuje permutacije f na stanje, a poziva se iz roditeljskog objekta Engine, koji je objekt srednjeg sloja, ukoliko još uvijek postoji neobrađeni dio poruke ili meta podataka. Funkcija Piston.GetTag dodaje poruci oznaku T.

GetTag(T, l):
assert ($l \leq Rs$)
T.put(state[i]) for $i \in \{0, \ldots, l-1\}$
Return

T- oznaka
l- bitovi stanja S
Rs- indeks kraja poruke

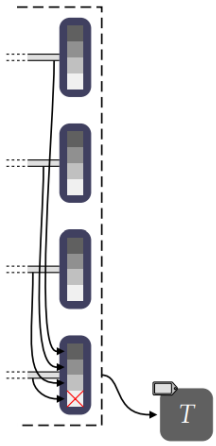
Pretpostavlja se da su ulazni niz poruka i meta podataka statični između inject funkcija i kriptiranja, tj. da nema nadopunjavanja poruka ili meta podataka. Engine je srednji sloj u ovoj arhitekturi. U ovom sloju ne održavaju se stanja već se oslanja na to da svaki Piston objekt održava stanja i offsete. Engine sloj ima 4 funkcije iz kojih pozivaju funkcije Piston objekata od kojih svaki obrađuje fragment ulazne poruke. Funkcija Engine.Spark poziva funkciju Piston.spark za svaki Piston objekt. Funkcija Engine.Wrap šalje ulazni niz Piston objektima te poziva Piston.Crypt nad svim Piston objektima te u izlazni stream O sprema odgovarajuće izlazne nizove. Funkcija Engine.InjectCollective(X, diversifyFlag) ubacuje iste meta podatke svim Piston objektima pozivajući Piston.Inject funkciju te se koristi za ubacivanje SUV (secret and unique value) i ulančanih vrijednosti. Ako je diversifyFlag =

true, u meta podatke ubacuju se 2 bita: jedan koji enkodira stupanj paralelizma za separaciju domene između instanci s različitim brojem Piston objekata a drugi za enkodiranje indeksa Piston objekta za separaciju među Piston objektima i za izbjegavanje korištenja identičnog ključa. Funkcija Engine.getTag(T, l) poziva Piston.Spark nad svakim Piston objektom te skuplja odgovarajuće oznake u izlazni niz T pozivajući funkciju Piston.GetTag.

GetTags(T, l):
assert (if endOfMessage)
Spark(true, l)
Pistons[i].GetTag(T, $l[i]$) for $i \in \{0, . . ,П- 1\}$
Return

T- oznaka
l- bitovi stanja S
i- indeks piston objekta, $i \in \{0, . . ,П- 1\}$

Parametar l je zapravo vektor što omogućuje uzimanje različitog broja bitova za svaki Piston objekt. Proces završava pozivanjem funkcije Piston.Spark ili Piston.GetTag koje se mogu pozivati jedna za drugom ili paralelno. Funkcije Piston.Crypt i Piston.Inject također se mogu pozivati jedna za drugom ili paralelno. Motorist je najviši sloj u ovoj arhitekturi te on implementira korisničko sučelje. Motorist kontrolira Engine objekte te poziva parametrizirane Piston objekte. Sam Motorist parametriziran je jedinicom poravnavanja W što omogućuje da su fragment start offsets i duljina oznaka T, ulančane vrijednosti i fragmenti višekratnici W dopustajući manipuliranje podacima u višebajtnim komadima. Stanja se odražavaju atributom phase koji može biti: Ready- Motorist objekt je inicijaliziran i još uvijek nema ulaznih podataka, Riding- Motorist objekt je procesuirao SUV i u stanju je zaprakirati/ otpakirati podatke i Failed- Motorist objekt je primio neispravnu oznaku T. Sesija započinje pozivom funkcije Motorist.StartEngine kada je Motorist objekt u stanju *ready* te se koliko god je puta to potrebno poziva Motorist.Wrap funkcija. Funkcija Motorist.StartEngine(SUV, tagFlag, T, decryptFlag) započinje sesiju čitanjem SUV iz ulaznog niza. Zastavica forgetFlag označuje je li čvor (knot) potreban ili ne, ukoliko je zastavica decryptFlag postavljena na 0 vraća se oznaka T a u protivnom se verificira oznaka T i ako je verifikacija oznake T bila uspješna prelazi se u stanje *riding*. Funkcija Motorist.MakeKnot omogućuje da stanja Piston objekata međusobno ovise te se poziva samo u slučaju kada je to potrebno.



Slika 3 Slika prikazuje čvor. U ovom slučaju međusobno ovise stanja 4 Piston objekta. Svaka linija prikazuje stanje Piston objekta. Ulančane vrijednosti iz svih Piston objekata zajedno su ubačene u svaki Piston objekt.

U stanju *riding* poziva se i funkcija Motorist.Wrap(I, O, A, decryptFlag, forgetFlag) kojom se kriptogram zapakirava kada je decryptFlag = 0 pri čemu I predstavlja ulaznu poruku, a O izlaznu enkriptiranu poruku ili otpakirava kada je decryptFlage = 1 pri čemu je I ulazni niz koji predstavlja enkriptiranu poruku, a O izlazni niz koji sadrži dekriptiranu poruku ukoliko je oznaka T ispravna. Obje strane koje komuniciraju moraju sinkronizirati vrijednosti parametara tagFlag i forgetFlag.

KEYAK je familija parametriziranih algoritama za autentificiranu enkripciju. Svi algoritmi koriste

KECCAK- p permutacije i imaju Motorist modul sa spužvastom konstrukcijom. Paket ključa koristi jednostavan padding te se sastoji od 3 dijela: prvi bajt koji označava duljinu ključa u bajtovima, ključa te jednostavnog paddinga. Keyak algoritmi imaju unificirani način enkodiranja tajnog ključa kao prefiks od SUV-a.Instanca Keyak objekta definira se:

Keyak $[b, n_r, \Pi, c, \tau]$ = *Motorist* $[Keccak_p [b, n_r], \Pi, W, c, \tau]$

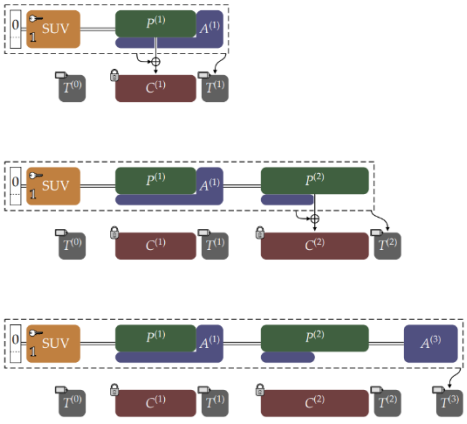
gdje je: W = $max(b/25 , 8)$.

Postoji 5 instanci Keyak familije algoritama:

NAZIV	b	Broj paralelnih Piston objekata
River Keyak	800	1
Lake Keyak	1600	1
Sea Keyak	1600	2
Ocean Keyak	1600	4
Lunear Keyak	1600	8

Preporuča se korištenje Lake Keyak instance. Za River Keyak W = 32 i paket ključa duljine je l = 36 bajtova, dok je za sve preostale instance W = 64, paket ključa je 40 bajtova. KEYAK algoritmi podupiru sesije u kojima cijeli niz poruka može biti autentificiran. Sesija je inicijalizirana učitavanjem ključa, noncea i oznake T za svaku poruku koja autentificira cijeli niz poruka prije nje. Tijekom sesije svi objekti koji međusobno komuniciraju moraju održavati stanje. Sve instance koriste javni broj poruke varijabilne duljine ili nuncij N, ali ne koriste privatni broj poruke. Poruka se može sastojati i samo od meta podataka te zbog toga nakon kriptiranja neće biti kriptirane poruke nego samo meta podaci i oznaka T. Zamotanje obuhvaća zamotavanje svake poruke u kriptogram i enkripciju u kriptiranu poruku te računanje autentificirajuće oznake T za svaki niz poruka. Suprotno tome, odmotavanje obuhvaća dekripciju kriptirane poruke i verifikaciju autentificirajuće oznake T i vraćanje dekriptirane poruke ukoliko je oznaka T validna ili praznog stringa ako oznaka T nije validna.

KEYAK algoritmi podupiru sesije u kojima se autentificira cijeli niz poruke te obzirom na to da su sesije podržane smanjuje se potreba za nonceom. Sigurnost algoritma se bazira na tajnosti unutarnjih stanja Piston objekata čime se sprječavaju tzv. side channel napadi. Korištenje rundi i KECCAK- p funkcija također štiti od side channel napada jer se lako mogu implementirati u stvarnom vremenu. Mehanizam „zaboravljanja“ u entitetu Motorist osigurava tajnost podataka jer iako bi se napadom moglo saznati stanje objekta, ne može se nikako znati stanje prije što bi bilo ključno za otkrivanje podataka. Unatoč zlouporabe noncea, autentifikacija je osigurana te je šteta privatnosti limitirana. Omogućene su paralelna enkripcija ili dekripcija paralelnim korištenjem različitih Piston objekata.



Slika 4 prikazuje sesiju u Motorist modu. Sesija započinje danom SUV, a zatim se procesuiraju poruka P1 i meta podaci A1

paralelno. Poruka je enkriptirana u C1, a T1 autentificira SUV, P1 i A1. Nakon procesuiranja sljedeće poruke T2 autentificira SUV, P1, A1, P2 i A2 te nakon procesuiranja treće poruke T3 autentificira cijeli sesiju, tj. SUV, P1, A1, P2, A2, P3 i A3 pri čemu je u ovom slučaju P3 prazna poruka.

KEYAK algoritmi pružaju bolje hardware performanse u usporedbi s drugim algoritmima poput AES- GCM.

KEYAK algoritmi pogodni su za autentificiranu enkripciju niza poruka. Same poruke često se prirodno grupiraju u sesiju primjerice na internetskoj vezi, prilikom transakcije pomoću pametnih kartica ili u razgovoru u nekoj aplikaciji stoga je praktično kriptirati te poruke upravo jednim od KEYAK algoritama. Motorist modul implementiran u KEYAK algoritmima može se koristiti za sigurnu dvosmjernu komunikaciju, ali mora se znati tko je pošiljatelj svake poruke što se može jednostavno ostvariti stavljanjem pošiljateljskog jedinstvenog identifikacijskog broja u meta podatke. KEYAK algoritmi dizajnirani su za uređaje s visokim performansama.

3. Programsko rješenje

3.1 Problem koji naš program rješava

Svrha našeg programa je olakšanje demonstracije rada odabranih algoritama. Korisnik s lakoćom može odabrati algoritam, unijeti ili generirati parametre i šifrirati ili dešifrirati blok teksta te je zbog toga idealan za edukativne svrhe. Također, modularan dizajn aplikacije dozvoljava lagano proširenje dodatnim algoritmima

3.2 Tehničke značajke programa

Programska podrška je ostvarena pomoću radnog okvira Qt (verzija 5) i jezika C++. Radni okvir Qt je odabran zbog lakoće rada i interoperabilnosti jezika C++ (Qt5, programsko sučelje prema algoritmima) i C (implementacija algoritama).

Programsko sučelje šifriranja i dešifriranja je ostvareno standardnim programskim jezikom C++ i nalazi se u mapi *Cryptography*.

Pomoćne funkcije za rad s datotekama i enkodiranje iz i u *base64* format se nalaze u mapi *Util*.
Korisničko sučelje se nalazi u datoteci *MainWindow(.h/.cpp)*.

3.3 Opis programa

Program omogućuje korisniku odabir algoritma kojim želi kriptirati određenu poruku ili datoteku. Korisnik pomoću grafičkog sučelja odabire želi li kriptirati ili dekiptirati ulaznu poruku ili datoteku, algoritam enkripcije ili dekripcije te bira želi li unijeti ručno parametre ili će iskoristiti defaultne.

3.4 Upute za pokretanje programa

Program se može instalirati na Windows operacijskom sustavu. S web stranice projekta treba se skinuti .zip datoteka u kojoj se nalazi program i raspakirati.

3.5 Upute za korištenje

Korisničko sučelje programske potpore se sastoji od 4 glavna dijela:

1. Dijela za odabir algoritma
2. Dijela za unos poruke te ispis izlaza algoritma
3. Dijela za unos parametara
4. Dijela za pokretanje algoritma te prikaza uputa za korištenje

Odabir algoritma

Pritiskom na dio za odabir algoritma otvara se padajući izbornik koji omogućuje odabir algoritma. Odabirom pojedinog algoritma, polja "Ključ", "AD", I "NPUB" se postavljaju na početne vrijednosti.

Ulaz i izlaz

Na početku, programska podrška se nalazi u stanju za šifriranje (naznačeno gumbom koji ima strelicu prema desno i tekstom "Šifriranje"). Korisnik unosi tekst u tekstualno polje s lijeve strane nazvano "Tekst" ili odabire datoteku za šifriranje (pritiskom gumba "Odaberi" ili unosom relativne ili apsolutne putanje datoteke u tekstualno polje za datoteku). Izlaz algoritma nakon pokretanja se prikazuje na desnoj strani.

U slučaju da se korisnik želi prebaciti u način za dešifriranje, pritišće gumb sa strelicom prema desno i tekstom "Šifriranje". Nakon toga se gumb mijenja smjer strelice prema lijevo i tekst u "Dešifriranje" što označuje da se programska potpora nalazi u načinu za dešifriranje. Korisnik sada na jednak način kao prije unosi poruku u tekstualno polje s desne strane "Tekst" ili odabire datoteku za šifriranje s desne strane. Izlaz algoritma se prikazuje na lijevoj strani.

Parametri

Svaki algoritam prima 3 parametra

1. Ključ
2. AD
3. NPUB

Duljine parametara (u bajtovima) su specificirane u zagradi iza imena parametra. Npr. algoritam Acorn-128 prima ključ duljine 16 bajta, AD duljine 16 bajta i NPUB duljine 16 bajta.
Svi parametri se mogu unositi ručno ili se mogu automatski generirati. Automatsko generiranje generira parametar tražene duljine i nije potrebna korisnička intervencija.
Za ključ također postoji opcija odabira tekstualne datoteke koja sadrži ključ tražene duljine.
U slučaju da duljina parametra ne odgovara traženoj duljini za pojedini algoritam, programska potpora daje prozor s informacijom o greški (neispravnost duljine parametra).

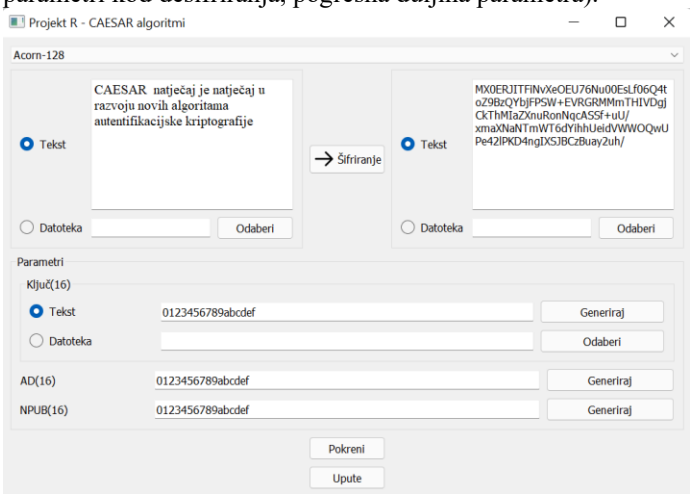
Pretpostavljeni parametri prilikom odabira algoritma

Svaki algoritam prilikom inicijalnog odabira ima postavljene pretpostavljene parametre tražene duljine. U tablici su navedeni pretpostavljeni parametri.

	Ključ	AD	NPUB
Acorn-128	0123456789abcdef	0123456789abcdef	0123456789abcdef
Aegis-128	0123456789abcdef	0123456789abcdef	0123456789abcdef
Ascon-128	0123456789abcdef	0123456789abcdef	0123456789abcdef
Deoxys-II-128	0123456789abcdef	0123456789abcdef	0123456789abcde
KetjeSrV2	0123456789abcdef	0123456789abcdef	0123456789abcdef0123456789abcde

Pokretanje i upute

Pritiskom na gumb "Pokreni", algoritam šifrira ili dešifrira poruku (ovisno o načinu rada programske potpore) i ispisuje izlaz ili daje prozor s informacijom o greški (npr. nema dovoljno slobodne memorije, korišteni su krivi parametri kod dešifriranja, pogrešna duljina parametra).



Slika 5 prikazuje primjer korištenja programa: odabrani algoritam je Acorn- 128, poruka koja se treba šifrirati nalazi se u polju za tekst s lijeve strane, postavljeni su pretpostavljeni parametri, odabrana je opcija šifriranje te se šifrirana poruka nalazi u polju za tekst s desne strane.

4. Literatura

1. Guido Bertoni, Joan Daemen, Michael Petters, Gilles van Assche, Ronny van Keer (2016). CAESAR submission: KETJE v2
2. Guido Bertoni, Joan Daemen, Michael Petters, Gilles van Assche, Ronny van Keer (2016). CAESAR submission: KEYAK v2
3. Hongjun Wu , Bart Preneel (2016). AEGIS: A Fast Authenticated Encryption Algorithm (v1.1)
4. Hongjun Wu (2016). ACORN: A Lightweight Authenticated Cipher (v3)
5. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, Martin Schlöaffer (2016). Ascon v1.2 Submission to the CAESAR Competition
6. Jérémy Jean , Ivica Nikolić , Thomas Peyrin , Yannick Seurin (2016). Deoxys v1.41
7. (2014). Cryptographic competitions- Introduction, dostupno na: https://competitions.cr.yp.to/index.html
8. (2014). Cryptographic competitions- Secret- key Cryptography, dostupno na: https://competitions.cr.yp.to/secret.html
9. (2021). Rijndael S- box, dostupno na: https://en.wikipedia.org/wiki/Rijndael_S-box