Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.3
DOI: https://doi.org/10.24867/10BE06Negulic

JEDNO REŠENJE PROGRAMSKE PODRŠKE ZA BEZBEDNO ČUVANJE PODATAKA U AUTOMOBILSKOJ INDUSTRIJI

ONE SOLUTION FOR PERSISTENT DATA STORAGE IN AUTOMOTIVE INDUSTRY

Ivan Negulić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu biće opisan dizajn i implementacija rešenja generičkog bezbednog sistema za trajno čuvanje podataka u programskom jeziku C, u AUTOSAR okruženju. Takođe, implementirane su metode zaštite podataka, algoritmima simetričnog, AES i asimetričnog RSA kriptovanja.*

Ključne reči: *Generički bezbedan sistem, kriptografija*

Abstract – *This paper will describe the design and implementation of a generic secure system solution for persistent data storage in the C programming language, in the AUTOSAR environment. Also, data protection methods, symmetric, AES and asymmetric, RSA, encryption algorithms have been implemented.*

Keywords: *Generic secure system, cryptography*

1. UVOD

Svakodnevna upotreba vozila zahteva unapređenje postojećih usluga, u smislu bezbednosti, sigurnosti i udobnosti vožnje, obezbeđivanju multimedijalnih sadržaja i slično. Sve ovo zahteva od kompanija koje se bave proizvodnjom automobilske opreme i vozila da stalno unapređuju svoje proizvode i usluge koje nude na svojim vozilima i uređajima. Stalno unapređenje fizičke arhitekture računara i sistemske programske podrške, kao i sistema zasnovanih na računaru obezbeđuje konstantan razvoj funkcionalnosti, različitih nivoa kompleksnosti, kao sto su elektronska kontrola stabilnosti, preciznije doziranje smeše za sagorevanje u cilindrima motora sa unutrašnjim sagorevanjem, koje obezbeđuje smanjenje potrošnje goriva, praćenje potrošnje električne energije baterija u vozilima na električni pogon, pa sve do auto pilota i autonomne vožnje. Računarski sistemi koji se koriste u automobilskoj industriji, odnosno u automobilskim sistemima, po performansama ne zaostaju za savremenim PC računarima. Modernim automobilima se postavljaju visoki zahtevi vezani za sigurnost, ekonomičnost i udobnost [1]. Jedan od najvećih izazova sa inženjerske tačke gledišta predstavlja prenos podataka, skladištenje i manipulacija podacima.

Trajno održanje sadržaja memorije u automobilskim sistemima može se posmatrati kao funkcija za praćenje pređene kilometraže, prosečne potrošnje goriva ili prosečne potrošnje električne energije u vozilima na električni pogon, pamćenje omiljenih radio stanica položaj sedišta, volana, retrovizora u zavisnosti od trenutnog vozača, do bezbednosnih kritičnih podataka, kao što su stanje elektronske parking kočnice, slike sa kamere, radara, i ostalih senzora u trenucima pre udesa automobila (crna kutija).

Razvoj programske podrške u automobilskim sistemima zahteva razvoj po najvišim bezbednosnim standardima, radi obezbeđivanja zahtevanog ponašanja sistema, ranog otkrivanja kvarova, smanjujući mogućnost nepredviđenog ponašanja sistema na prihvatljiv nivo.

2. TEORIJSKE OSNOVE

Softverski model - Softverski proces je skup aktivnosti i dobijenih rezultata čiji je cilj razvoj i evolucija softvera. Osnovne aktivnosti unutar softverskog procesa su: specifikacija, modelovanje, implementacija, verifikacija, i validacija. Softverski model je idealizovan prikaz softverskog procesa, kojim se određuje željeni način odvijanja i međusobnog povezivanja aktivnosti, odnosno model je opis sistema napisan korišćenjem jezika koji podržava automatsku interpretaciju od strane računara. Svi softverski modeli su okarakterisani sledećim aktivnostima:

- Specifikacija – analiza zahteva korisnika, utvrđivanje šta softver treba da radi.
- Modelovanje – definiše se sistem, način rada komponenti, komunikacija između komponenti, odnosno uprošćeno, projektuje se rešenje koje određuje kako će softver raditi.
- Implementacija – modelovani sistem realizuje se pomoću programskih jezika i alata.
- Verifikacija i validacija – proverava se da li softver radi prema zadatoj specifikaciji korisnika. Ovaj korak se svodi na testiranje, ali postoje i druge metode.
- Održavanje, evolucija – nakon uvođenja u upotrebu, softver se dalje popravlja, menja, nadograđuje, sve u skladu sa zahtevima korisnika.

2.1. Modeli i metamodeli

Model je apstrakcija sistema koji se često koristi za zamenu sistema koji se proučava. Model predstavlja delimični i pojednostavljeni prikaz sistema, pa je obično potrebno stvaranje više modela da bi se jasnije predstavio i razumeo sistem koji se proučava. Modeliranje je dobro poznata tehnika usvojena od strane inženjerskih oblasti, kao i drugih oblasti poput fizike, matematike, ekonomije i drugih. Ovde se fokusiramo na modele u softverskom inženjerstvu i informacionim sistemima. Ovde ćemo predstaviti suštinske pojmove na kojima se zasniva inženjersko upravljanje modelima (MDE – Model-Driven

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Ilija Bašičević, red. prof.

Engineering), odnosno koncept sistema, modela, meta-modela i njihovih odnosa. U kontekstu MDE, definišemo "sistem kao generički koncept za označavanje softverske aplikacije, softverske platforme ili bilo kog drugom softverskog artefakta". Osnovna ideja je razmatranje modela na visokom nivou apstrakcije, bez detalja implementacije. Podizanjem nivoa apstrakcije smanjuje se složenost modela. Povećanje produktivnosti se postiže maksimiziranjem kompatibilnosti između sistema, korišćenjem standardizovanih modela, uprošćavanjem procesa dizajniranja, olakšavanjem komunikacije između različitih timova koji rade na sistemu, korišćenjem standardizovane terminologije, i korišćenjem tehnika i tehnologija koje su se pokazale kao najbolje u praksi (*Best practice*). Obećavajući pristup za rešavanje problema kompleksnih fizičkih arhitektura i programskih podrški, usled nemogućnosti programskih jezika treće generacije da smanje kompleksnost, i izraze koncepte problema određenih domena, je razvoj programske podrške korišćenjem inženjerstva zasnovanog na modelima [2].

2.2. Generator koda

U računarskoj nauci, termin automatsko programiranje identifikuje vrstu računarskog programiranja u kojem neki mehanizam generiše računarski program kako bi omogućio ljudskim programerima da napišu kod na višem nivou apstrakcije [3]. Jedan od najranijih programa koji se može prepoznati kao prevodilac nazvao se Autocode. Parnas je zaključio da je "automatsko programiranje uvek bilo eufemizam za programiranje na jeziku višeg nivoa nego što je tada bio dostupan programeru" [4]. Pod generisanjem programskog koda u ovom radu misli se na programske alate koji se koriste za obradu šablona. Upotreba ovakvih alata je korisna u slučajevima gde se slični ili isti delovi koda pojavljuju više puta, ili kada se zahtevi i specifikacije često menjaju. U slučaju promene specifikacije, neophodno je samo ponovo generisati kod. Ovakav način rada znatno smanjuje vreme za pisanje koda, povećava čitljivost koda, podstiče organizaciju izvornog koda u operativno različite slojeve, povećava produktivnost.

2.3. Trajno čuvanje memorije

U računaru se memorija odnosi na uređaj koji se koristi za čuvanje informacija za trenutnu upotrebu u računaru ili srodnom hardveru [4]. Prema trajnosti podataka po nestanku napajanja prepoznajemo dve vrste memorija: promenljive (volatile) i nepromenljive (non-volatile) memorije. Nepromenljive memorije ili memorije sa stalnim sadržajem su memorije koje mogu da uskladištite informacije i po nestanku napajanja.

2.4. CRC

Ciklična provera redundantnosti (CRC – Cyclic Redudance Check) je algoritam za otkrivanje grešaka koji se obično koristi u digitalnim mrežama i uređajima za skladištenje podataka, da bi se otkrile slučajne promene neobrađenih podataka. Blokovi podataka koji ulaze u ove sisteme dobijaju priložene kratke kontrolne vrednosti, zasnovane na ostatku deljenja polinoma sadržaja blokova podataka. Ciklična provera redundantnosti (CRC) zasniva se na deljenju u prstenu polinoma preko konačnog polja, odnosno Galovog polja GF(2). Ako posmatramo niz bitova kao $a_n a_{n-1} a_{n-1} \ldots a_1 a_0$, onda taj niz odgovara polinomu $M(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x + a_0$. Izračunavanje CRC odgovara Euklidskom deljenju polinoma nad Galovim poljem GF(2):

$$M(x) \cdot x^n = Q(x) \cdot G(x) + R(x) \qquad (1)$$

Gde je M(x) originalni polinom poruke, a G(x) je polinom generator stepena n. $M(x) \cdot x^n$ je originalna poruka na kojoj je dodato n nula na kraju, Q(x) je polinom koji nas ne zanima, a R(x) je preostali polinom koji predstavlja CRC kontrolnu sumu. Korišćenjem operacije ostatka pri deljenju, odnosno operacije modula, može se reći da je:

$$R(x)\ M(x) \cdot x^n \bmod G(x) \qquad (2)$$

Koeficijenti ostatka deljenja dodaju se na kraj poruke. Ostatak je vrednost CRC koja se zajedno sa ulaznim podacima šalje prijemniku, koji ponovnim računanjem CRC primljenih podataka, koji upoređuje izračunatu CRC sumu i primljenu CRC sumu. Ako je vrednost CRC nula, tada prilikom prenosa verovatno ne nastaje bitna greška, a u slučaju da je vrednost različita od nule u prenosu sigurno postoji greška.

2.5. AES

Napredni standard šifriranja (AES), poznat i po originalnom imenu Rijndael [5] je specifikacija za šifrovanje elektronskih podataka koju je ustanovio Nacionalni institut za standarde i tehnologiju (NIST) u 2001. godini [6].

Specifikacija algoritma

AES je zasnovan na principu dizajna poznatom kao supstituciona i permutaciona mreža i efikasan je kako u softveru, tako i u hardveru. AES je varijanta Rijndela koji ima fiksnu veličinu bloka od 128 bita i veličinu ključa od 128, 192 ili 256 bita. Veličina ključa koji se koristi za AES šifru određuje broj krugova transformacije koji pretvaraju ulaz, koji se naziva *plaintext,* u konačni izlaz, koji se naziva *chipertext*. Broj rundi je:

- 10 rundi za 128-bitni ključ,
- 12 rundi za 192-bitni ključ,
- 14 rundi za 256-bitni ključ.

Svaki krug se sastoji od nekoliko koraka obrade, uključujući i ona koji zavisi od samog šifriranja. Skup obrnutih zaokruživanja primenjuje se za transformisanje šifriranog teksta nazad u originalni tekst koristeći isti ključ za šifriranje.

AES algoritam je blok-šifra koja koristi isti enkripcijski ključ u procesu šifrovanja i dešifrovanja podataka koja se primenjuje u nekoliko iteracija (rundi). AES ima 10 rundi (može imati 10, 12 ili 14 rundi u zavisnosti od dužine ključa), a svaka runda se sastoji od 4 operacije koje se primenjuje nad elementima matrice dimenzije 4x4 bajta, a to su:

1. SubBytes (nelinearna zamena bajtova pomoću supstitucione tabele).
2. ShiftRows (promena mesta unutar bajtova iste kolone).
3. MixColumns (transformacija bajtova unutar iste kolone).
4. AddRoundKey (inicijalno dodavanje ključa).

2.6. RSA

RSA (Rivest – Shamir – Adleman) je jedan od prvih kriptosistema sa javnim ključem i široko se koristi za siguran prenos podataka. U takvom kriptosistemu ključ za šifrovanje je javni i razlikuje se od ključa za dešifrovanje koji se čuva u tajnosti (privatni). U RSA se ta asimetrija zasniva na praktičnoj poteškoći faktorizacije proizvoda dva velika osnovna broja [7]. Kriptovanje se vrši po formuli:

$$C \equiv M^e \ (mod\ n) \qquad (3)$$

Dok se dekriptovanje vrši po formuli:

$$M \equiv C^d \ (mod\ n) \qquad (4)$$

Generisanje ključeva se vrši na sledeći način:

1. Biraju se dva cela prosta broja p i q.
   - Iz sigurnosnih razloga, celi brojevi p i q se biraju nasumično,
   - p i q se čuvaju u tajnosti.
2. Izračunava se

$$n = pq \qquad (5)$$

   - n se koristi kao modul i za javne i za privatne ključeve, njegova dužina je obično izražena u bitovima,
   - n se šalje kao deo javnog ključa.
3. Izračunava se Ojlerova funkcija

$$\varphi(n) = (p-1)*(q-1) \qquad (6)$$

   - $\varphi(n)$ se čuva u tajnosti.
4. Bira se ceo broj e tako da $1 < e < \varphi(n)$ i da zadovoljava $NZD(e, \varphi(n)) = 1$
   - e se šalje kao deo javnog ključa.
5. Izračunava se d kao

$$d \equiv e^{-1} \ (mod\ \varphi(n)) \qquad (7)$$

   - d se čuva kao u tajnosti kao eksponent privatnog ključa.

3. KONCEPT REŠENJA

Ovo poglavlje prikazuje koncept rešenja primenjen u ovom radu. Projekat se sastoji iz dva dela: ručno pisan deo programskog koda, i generisani deo programskog koda. Ručno pisani deo implementacije sadrži implementaciju koja je nezavisna od ulaznih podataka, dok implementacija u generisanom delu direktno zavisi od ulaznih podataka. Ovakva realizacija se javlja, jer smanjuje greške u pisanju koda, koju se u ovom slučaju generiše, a takođe je primećeno dobro ponašanje u radu na komercijalnim projektima.

3.1. Negenerisani deo

Negenerisani deo će obezbeđivati funkcionalnost očuvanja blokova podataka, koristeći stalne memorije. U ovom delu će biti implementirane funkcije koje obezbeđuju čitanje i pisanje podataka glavne i pomoćne datoteke blokova podataka u stalnoj memoriji. Kako bismo obezbedili siguran proces čitanja i pisanja blokova podataka koriće se dve datoteke glavna i pomoćna datoteka za svaku softversku komponentu u sistemu. Glavna datoteka obezbeđuje čitanje i skladištenje svih blokova podataka, dok pomoćna datoteka, takođe obezbeđuje ovu funkcionalnost, s tim da će se ona koristiti u slučaju da je glavna datoteka oštećena, ili ako je utvrđeno da podaci u glavnoj datoteci nisu validni, odnosno da je integritet podataka narušen. Integritet se može narušiti u slučaju kvara stalne memorije, sistemske greške prilikom čitanja ili pisana u stalnu memoriju, ili u slučaju neovlašćenog pristupa memoriji i promeni sadržaja memorije. Bezbednost pomoćne i glavne datoteke je obezbeđena kriptografskim metodama AES i RSA zaštite podataka, a očuvanje integriteta obezbeđuje algoritam za izračunavanje CRC vrednosti, gde se proverava, nakon uspešnog dekriptovanja podataka iz datoteke, ispravnost CRC vrednosti.

3.2. Generisani deo

Generisani deo obezbeđuje funkcije koje generišu izvorni programski kod na osnovu ulaznih datoteka. U ovom slučaju ulazna datoteka je u XML formatu (eXtensible Markup Language). Ukoliko ulazna datoteka nije XML formata prijavljuje se greška, i generisanje programskog koda se prekida. Generisani deo programskog rešenja obezbeđuje funkcije trajnog čuvanja sadržaja memorije celom AUTOSAR sistemu. Funkcionalnosti koje se obezbeđuju sistemu se mogu podeliti u dve grupe: trajno čuvanje sadržaja memorije pre gašenja uređaja, i povratak sadržaja memorije po uključenju. Ovakav pristup omogućava iterativni i inkrementalni razvoj programske podrške. U slučaju promene ulaznih parametara, dovoljno je samo ponovo izgenerisati deo programske podrške koji zavisi od ulaznih parametara modela, bez potreba dodatnih ručnih izmena u programskom kodu. Na ovaj način se omogućava brz odgovor na promenu ulaznih podataka (XML datoteke).

4. VALIDACIJA I VERIFIKACIJA

Sastavni deo svake programske podrške su zahtevi za čestom promenom specifikacije zahteva. Specifikacija zahteva koja je na početku zahtevana sklona je čestim promenama, i samim tim potrebne su odgovarajuće promene u izvornom programskom kodu. Ove promene se zahtevaju kako bi programsko rešenje dovelo do stabilnijeg rada u praksi. U praksi česte promene, izmene procesa i dodavanje novih funkcionalnosti je čest uzrok nastanka grešaka i nepouzdanog izvršavanja programske podrške. Iz svih ovde navedenih razloga bitna stavka u razvoju programske podrške zahteva određena testiranja kako bi se utvrdila ispravnost funkcionalnosti programskog rešenja i potvrdila da zadovoljava specifikacije zahteva. Tokom izrade rešenja ispitivanje je vršeno na platformi sa Nvidia Tegra K1 procesorom[8], sa Linux distribucijom Jetson TK1 R21.5 i kernel verzijom 3.10.40 [9]. Tokom ispitivanja programskog rešenja dolazi se do zaključka da je u slučaju validnih ulaznih parametara, generisani izvorni C kod, kog generiše kod generator, ispravan i da ga je moguće kompajlirati. Na ovom rešenju je izvršeno pet grupa testova: grupa testova koja verifikuje ispravno čitanje podataka iz stalne memorije, grupa testova koja verifikuje ispravno upisivanje podataka pročitanih sa RTE

magistrale u stalnu memoriju, grupa testova koji verifikuju detekciju greške nastalu u stalnoj memoriji, grupa testova koja verifikuje ispravnost pročitanih podataka koji su kriptovani RSA algoritmom, grupa testova koja verifikuje ispravnost pročitanih podataka koji su kriptovani AES algoritmom. Svih pet grupa testova je prikazala da testovi u potpunosti zadovoljavaju zadatu specifikaciju.

5. ZAKLJUČAK

Prikazano rešenje programske podrške za trajno čuvanje sadržaja stalne memorije je razvijeno po AUTOSAR i MISRA C:2004 standardima, i se može lako integrisati u AUTOSAR projekat, ili kao deo posebnog ECU-a. Rešenje je moguće nadograditi dodatnim funkcionalnostima u zavisnosti od potrebe. Dodatne funkcionalnosti je potrebno implementirati u negenerisanom delu programske podrške. Sve promene u generisanom delu zavise isključivo od promene ulaznih podataka. U slučaju razdvajanja programskog koda na generisani deo i na deo koji se piše ručno (negenerisani), omogućena je brza i laka promena izvornog koda na osnovu ulaznih parametara, kao i podrška za iterativni i inkrementalni razvoj programske podrške. Kako bi se obezbedilo generisanje koda koji je moguće kompajlirati (prevesti na mašinski jezik) generator koda proverava validnost ulaznih podataka i parametara. Razvoj ovog rešenja u budućnosti može ići u pravcu implementacije dodatnih funkcija kao što su dijagnostičke informacije o stanju memorijskog uređaja, brojač upisa u memorijski uređaj i slično. Stalni napredak u automobilskoj industriji dovodi do toga da se u modernom vozilu svake sekunde stvara i obrađuje velika količina podataka, pa je neminovno da je potreba za čuvanjem i obezbeđivanjem podataka jedna od primarnih zadataka inženjera, kao što je i demonstrirano u ovom rešenju.

6. LITERATURA

[1] M. Jovanović *et al*, "Programska implementacija sistema za prikupljanje i obradu signala sa elektronskih kontrolnih jedinica u automobilskoj mreži," Telekomunikacioni forum TELFOR, Beograd, Srbija, 2015.

[2] D.C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," IEEE Computer, vol 39, no., pp. 25-31, Feb. 2006.

[3] Ricardo Aler Mur, "Automatic Inductive Programming Archived 2016-03-04 at the Wayback Machine", *ICML 2006 Tutorial*. June 2006.

[4] D. L. Parnas. "Software Aspects of Strategic Defense Systems." *American Scientist*. November 1985.

[5] *Daemen, Joan; Rijmen, Vincent (March 9, 2003).* "AES Proposal: Rijndael" *(PDF). National Institute of Standards and Technology. p. 1.* Archived *(PDF) from the original on 5 March 2013. Retrieved 21 February 2013.*

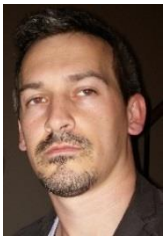
[6] "Announcing the ADVANCED ENCRYPTION STANDARD (AES)" *(PDF). Federal Information Processing Standards Publication 197. United States National Institute of Standards and Technology (NIST). November 26, 2001.* Archived *(PDF) from the original on March 12, 2017. Retrieved October 2, 2012.*

[7] *Rivest, R.; Shamir, A.; Adleman, L. (February 1978).* "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" *(PDF).* Communications of the ACM. ***21*** *(2): 120–126.* CiteSeerX 10.1.1.607.2677. doi:10.1145/359340.359342.

[8] "Tegra K1 Next-Gen Mobile Processor", 2017. http://www.nvidia.com/object/tegra-k1-processor.html.

[9] "Linux For Tegra R21.5", 2017. https://developer.nvidia.com/linux-tegra-r215.

Kratka biografija:



Ivan Negulić rođen je u Novom Sadu 1983. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva - Računarska tehnika i računarske komunikacije odbranio je 2019. god.

kontakt: ivan.negulic@gmail.com