**Sadržaj za diplomski rad na temu mapiranja memorije u mikrokontroleru**

1. **Uvod:**
   * Objašnjenje osnovnih pojmova (mikrokontroler, memorija, adresni prostor).
   * Značaj mapiranja memorije u dizajnu embedded sistema.
2. **Pregled arhitekture mikrokontrolera:**
   * Opis različitih tipova memorije (ROM, RAM, EEPROM).
   * Adresni prostor i segmentacija memorije.
   * Primeri različitih arhitektura (ARM, AVR, PIC).
3. **Proces mapiranja memorije:**
   * Korišćenje linker skripti i map fajlova.
   * Podešavanje memorijskih registara.
   * Mapiranje periferija.
4. **Praktični primeri:**
   * Implementacija osnovnog sistema za mapiranje memorije.
   * Inicijalizacija i konfiguracija memorije u specifičnom mikrokontroleru.
   * Analiza rezultata i performansi.
5. **Zaključak:**
   * Sumiranje ključnih aspekata mapiranja memorije.
   * Mogućnosti za optimizaciju i dalji rad.
6. **Reference:**
   * Literatura i izvori koji pokrivaju arhitekturu mikrokontrolera i tehniku mapiranja memorije.

**Mapiranje memorije u mikrokontroleru**

Mapiranje memorije u mikrokontroleru je proces povezivanja fizičke memorije sa određenim adresnim prostorom, tako da CPU može pristupiti memoriji putem specifičnih adresa. Ovaj proces omogućava da mikrokontroler efikasno koristi dostupne resurse, kao što su RAM, ROM, EEPROM, i memorijski-mapa perifernih uređaja.

**Kako se radi mapiranje memorije?**

1. **Definisanje adresnog prostora:** Mikrokontroleri obično imaju unapred definisane adresne prostore za različite vrste memorije, kao što su RAM, ROM, i I/O portovi. Adresni prostor je podeljen na segmente, pri čemu svaki segment odgovara specifičnom tipu memorije ili periferiji.
2. **Podešavanje memorijskih registara:** Mikrokontroleri koriste specijalizovane registre za podešavanje memorijskih mapa. Ovi registri kontrolišu kako se određene memorijske adrese mapiraju na fizičku memoriju ili periferne uređaje.
3. **Inicijalizacija memorije:** Pri pokretanju sistema, bootloader ili inicijalizacioni kod postavlja osnovne mape memorije tako da softver može efikasno pristupiti resursima mikrokontrolera.
4. **Korišćenje linker skripti:** U C i Asembler programiranju, linker skripte definišu kako se sekcije koda i podataka smeštaju u memorijski prostor mikrokontrolera. Na primer, .text sekcija može biti smeštena u Flash memoriju, dok .data i .bss sekcije mogu biti smeštene u RAM.

**Definisanje sekcija u C programskom jeziku**

U C programskom jeziku, sekcije u koje se smeštaju promenljive i funkcije mogu se definisati korišćenjem specifičnih atributa. Na primer:

|  |
| --- |
| int my\_var \_\_attribute\_\_((section(".my\_section"))) = 0;  void my\_function() \_\_attribute\_\_((section(".my\_section"))) {  // Kod funkcije  } |

Ovim se promenljive i funkcije smeštaju u sekciju nazvanu .my\_section, koja se zatim može mapirati na specifičnu memorijsku lokaciju korišćenjem linker skripte.

**Vrste kompajlera i linkera za C i Asembler u embedded programiranju**

Postoji nekoliko vrsta kompajlera i linkera koji se koriste u embedded programiranju:

1. **Kompajleri:**
   * **GCC (GNU Compiler Collection):** Otvoreni kompajler koji podržava više arhitektura.
   * **Keil C:** Komercijalni kompajler specijalizovan za ARM mikrokontrolere.
   * **IAR Embedded Workbench:** Komercijalni kompajler sa podrškom za širok spektar mikrokontrolera.
   * **AVR-GCC:** Verzija GCC-a specijalizovana za AVR mikrokontrolere.
   * **SDCC (Small Device C Compiler):** Otvoreni kompajler za mikrokontrolere sa ograničenim resursima.
2. **Linkeri:**
   * **GNU Linker (ld):** Standardni linker za GCC koji koristi linker skripte za raspored sekcija.
   * **Keil Linker:** Proprietarni linker za ARM mikrokontrolere.
   * **IAR Linker:** Proprietarni linker koji je deo IAR Embedded Workbench alata.
   * **Linkeri za specifične arhitekture:** Neki mikrokontroleri dolaze sa sopstvenim linkerima optimizovanim za specifične arhitekture (npr. AVR).

Ovi alati su ključni za kontrolu kako se kod i podaci smeštaju u memorijski prostor mikrokontrolera, omogućavajući optimizaciju performansi i efikasnost korišćenja resursa.

**Kako CPU pristupa memoriji?**

CPU pristupa memoriji koristeći adresni prostor, u kojem svaka memorijska lokacija ima jedinstvenu adresu. Proces pristupa memoriji obuhvata nekoliko koraka:

1. **Generisanje adrese:** Kada CPU treba da pristupi određenoj memorijskoj lokaciji, prvo generiše adresu te lokacije. Ova adresa se može generisati na osnovu instrukcije koju CPU trenutno izvršava.
2. **Slanje adrese preko adresne magistrale:** Adresa se prenosi preko adresne magistrale (Address Bus) do memorijskog kontrolera, koji upravlja pristupom memoriji.
3. **Prenos podataka:** Memorijski kontroler identifikuje memorijsku lokaciju i vraća podatke (ako se radi o čitanju) ili smešta podatke (ako se radi o pisanju) putem podatkovne magistrale (Data Bus).
4. **Kontrola pristupa:** Kontrolna magistrala (Control Bus) prenosi signale koji upravljaju procesom pristupa, kao što su signali za čitanje (Read) i pisanje (Write).

**Šta CPU sadrži od hardverskih komponenti?**

CPU (Central Processing Unit) je srž svakog računarskog sistema i sadrži sledeće ključne hardverske komponente:

1. **ALU (Arithmetic Logic Unit):** Izvodi aritmetičke i logičke operacije kao što su sabiranje, oduzimanje, AND, OR, i slično.
2. **Kontrolna jedinica (Control Unit):** Upravljava izvršavanjem instrukcija, kontroliše tok podataka između različitih delova CPU-a i generiše signale potrebne za izvršenje instrukcija.
3. **Registri:**
   * **General-purpose registri:** Koriste se za privremeno skladištenje podataka i rezultata operacija.
   * **Programski brojač (Program Counter, PC):** Sadrži adresu sledeće instrukcije koja treba da se izvrši.
   * **Stack Pointer (SP):** Pokazuje na vrh steka u memoriji, koji se koristi za privremeno skladištenje podataka i povratnih adresa funkcija.
   * **Statusni registar (Flags Register):** Sadrži zastavice koje odražavaju rezultate prethodnih operacija (npr. nula, prenos, prekoračenje).
4. **Interne magistrale:** Unutar CPU-a postoje magistrale koje povezuju ALU, registre, i kontrolnu jedinicu, omogućavajući internu komunikaciju.
5. **Dekoder instrukcija:** Interpretira binarne kodove instrukcija koje CPU prima i generiše signale za njihovo izvršavanje.
6. **Keš memorija (Cache):** Brza memorija koja privremeno skladišti često korišćene podatke i instrukcije, smanjujući vreme pristupa podacima iz glavne memorije.

**Kakvi sve registri i kakve sve magistrale postoje?**

**Registri:**

1. **General-purpose registri:** Višenamenski registri koji se koriste za privremeno skladištenje operanda i rezultata aritmetičkih/logičkih operacija. Primeri su AX, BX, CX, DX u x86 arhitekturi.
2. **Specijalni registri:**
   * **Program Counter (PC):** Drži adresu sledeće instrukcije koja treba da se izvrši.
   * **Stack Pointer (SP):** Pokazuje na trenutnu lokaciju na steku.
   * **Instruction Register (IR):** Sadrži trenutnu instrukciju koja se izvršava.
   * **Statusni ili Flag registar:** Sadrži zastavice koje ukazuju na rezultat operacija, kao što su Carry, Zero, Sign, Overflow.
   * **Base Register (BR) i Index Register (IR):** Koriste se za adresiranje u složenim metodama adresiranja.

**Magistrale:**

1. **Adresna magistrala (Address Bus):** Prenosi adrese od CPU-a do memorije ili I/O uređaja. Magistrala je obično jednosmerna.
2. **Podatkovna magistrala (Data Bus):** Prenosi podatke između CPU-a, memorije i I/O uređaja. Magistrala može biti dvosmerna, omogućavajući i čitanje i pisanje podataka.
3. **Kontrolna magistrala (Control Bus):** Prenosi kontrolne signale, kao što su signali za čitanje/pisanje, prekide, i druge signale za upravljanje radom sistema.

**Kako CPU pristupa RAM-u, a kako ROM-u?**

**Pristup RAM-u (Random Access Memory):**

* **Čitanje:** CPU šalje adresu željene memorijske lokacije preko adresne magistrale do RAM-a. RAM vraća podatke sa te adrese nazad u CPU preko podatkovne magistrale.
* **Pisanje:** CPU šalje adresu zajedno sa podacima koje treba da upiše na tu memorijsku lokaciju. Kontrolna magistrala prenosi signal za pisanje, što omogućava RAM-u da zabeleži podatke na zadatu lokaciju.

RAM je volatilan, što znači da gubi podatke kada je isključen, ali omogućava brze operacije čitanja i pisanja.

**Pristup ROM-u (Read-Only Memory):**

* **Čitanje:** Slično kao i kod RAM-a, CPU šalje adresu preko adresne magistrale. ROM vraća podatke sa te adrese nazad u CPU, ali ROM ne dozvoljava pisanje podataka.

ROM je nevolatilan, što znači da zadržava podatke čak i kad je napajanje isključeno. Koristi se za smeštanje firmware-a i programa koji se retko menjaju, poput bootloader-a.

U sažetku, CPU pristupa memoriji koristeći adresnu, podatkovnu i kontrolnu magistralu, pri čemu različite vrste memorije (RAM i ROM) imaju specifične načine čitanja i pisanja podataka.

|  |
| --- |
|  |

# Mikrokontroler

# Memorija u mikrokontroleru

# Memorijske sekcije

# Algoritmi za proveru integriteta podataka

CRC (Cyclic Redundancy Check), **checksum**, i **SHA (Secure Hash Algorithm)** su algoritmi koji se koriste za proveru integriteta podataka, ali se razlikuju po načinu na koji funkcionišu i njihovoj složenosti. Evo objašnjenja za svaku od njih:

1. **CRC (Cyclic Redundancy Check)**: CRC je algoritam koji se koristi za otkrivanje grešaka u podacima. Često se koristi u digitalnim mrežama i medijima za skladištenje podataka. Algoritam uzima niz podataka i kroz polinomske operacije stvara kontrolnu sumu (checksum), koja se koristi za proveru integriteta tih podataka.
2. **Checksum (Kontrolna suma)**: Checksum je jednostavniji način za proveru integriteta podataka. Na primer, može se sabrati sve vrednosti u nizu podataka, a rezultat te sume koristi se za proveru. Ako se podaci promene, nova kontrolna suma neće se podudarati s originalnom.
3. **SHA (Secure Hash Algorithm)**: SHA je porodica kriptografskih algoritama koji se koriste za izračunavanje **heš vrednosti** podataka. Ovi algoritmi stvaraju jedinstven niz fiksne dužine koji se naziva heš, koji predstavlja sadržaj ulaznih podataka. SHA se najviše koristi u kriptografiji i sigurnosnim aplikacijama, kao što su digitalni potpisi i autentifikacija podataka. Poznate verzije uključuju **SHA-1**, **SHA-256**, **SHA-512**, i druge.

Da sumiramo:

* CRC se koristi za otkrivanje grešaka u podacima,
* Checksum je jednostavan način za proveru integriteta podataka,
* SHA algoritmi se koriste za kriptografsku zaštitu i proveru podataka putem heš vrednosti.

Zajedničko ime za sve ove algoritme je **algoritmi za proveru integriteta podataka**. Takođe se mogu nazivati i **algoritmi za detekciju grešaka** ili **algoritmi za verifikaciju podataka**.

U specifičnim kontekstima, ovi algoritmi mogu biti klasifikovani kao:

* **Heš funkcije** (kao što su SHA, MD5) – koje stvaraju heš vrednosti za proveru integriteta.
* **Algoritmi za detekciju grešaka** (kao što su CRC i checksum) – koji otkrivaju promene u podacima i proveravaju da li je došlo do grešaka.

Dakle, svi ti algoritmi spadaju pod širu kategoriju metoda za proveru integriteta ili verifikaciju podataka.

NOVI GEN:

Kada podaci putuju kroz mreže, prolaze kroz uređaje ili se skladište na diskovima, postoji šansa da dođe do grešaka ili neovlašćenih promena. Da bismo bili sigurni da podaci ostaju tačni i nepromenjeni, koriste se **algoritmi za proveru integriteta podataka**. Ovi algoritmi pomažu u otkrivanju da li su podaci oštećeni, promenjeni ili kompromitovani tokom prenosa ili skladištenja.

**Šta su algoritmi za proveru integriteta podataka?**

Algoritmi za proveru integriteta podataka su matematički postupci koji obrađuju ulazne podatke (kao što su fajlovi, paketi podataka ili nizovi bitova) i generišu sažetak tih podataka, koji se koristi za proveru da li su podaci ostali nepromenjeni. Ovaj sažetak, koji se često naziva **kontrolna suma** ili **heš**, može se porediti sa sažetkom originalnih podataka da bi se proverilo da li su podaci izmenjeni.

**Najčešći algoritmi za proveru integriteta**

Postoji nekoliko različitih vrsta algoritama za proveru integriteta, a svaki od njih ima različitu primenu i način funkcionisanja. Neki od najpoznatijih su:

1. **Checksum (Kontrolna suma)**:
   * **Opis**: Kontrolna suma je jednostavan način za proveru integriteta podataka. U suštini, to je zbir svih bajtova ili bitova u skupu podataka.
   * **Kako se računa**: Ako, na primer, imate niz podataka predstavljenih brojevima, jednostavno sabirate sve vrednosti da biste dobili kontrolnu sumu. Kada primalac podataka dobije taj niz, ponovo izračunava kontrolnu sumu. Ako se ta kontrolna suma ne podudara s originalnom, podaci su oštećeni.
   * **Gde se koristi**: Često se koristi u jednostavnim protokolima prenosa podataka, kao što su FTP ili rudimentarni protokoli u mrežnim aplikacijama.
2. **CRC (Cyclic Redundancy Check)**:
   * **Opis**: CRC je sofisticiranija metoda koja koristi polinomske operacije. Generiše se specifičan broj (CRC vrednost) na osnovu podataka, a ta vrednost služi za proveru grešaka.
   * **Kako se računa**: Podaci se tretiraju kao binarni brojevi koji se dele određenim polinomom. Ostatak te deljenja je CRC vrednost.
   * **Gde se koristi**: CRC se široko koristi u mrežnim protokolima (npr. Ethernet) i uređajima za skladištenje (npr. CD, DVD) gde je važno brzo detektovati greške u podacima.
3. **SHA (Secure Hash Algorithm)**:
   * **Opis**: SHA je porodica kriptografskih algoritama koji kreiraju jedinstvenu heš vrednost za set podataka. Ovi algoritmi su daleko kompleksniji i pružaju visoku sigurnost.
   * **Kako se računa**: Algoritam uzima ulazne podatke, bez obzira na njihovu veličinu, i pretvara ih u heš vrednost fiksne veličine (na primer, SHA-256 uvek generiše heš od 256 bita). Heš vrednost je jedinstvena za svaki set podataka, i veoma je teško (praktično nemoguće) pronaći dva različita skupa podataka koji generišu istu heš vrednost.
   * **Gde se koristi**: SHA algoritmi se koriste u kriptografiji, za digitalne potpise, autentifikaciju, zaštitu fajlova i u blockchain tehnologijama. Verzije uključuju SHA-1, SHA-256, i SHA-512, od kojih se SHA-256 koristi u mnogim sigurnosnim aplikacijama, uključujući Bitcoin.
4. **MD5 (Message Digest Algorithm 5)**:
   * **Opis**: MD5 je još jedan kriptografski heš algoritam koji stvara heš vrednost od 128 bita.
   * **Kako se računa**: Slično SHA algoritmima, MD5 uzima ulazne podatke i stvara heš vrednost fiksne veličine. Međutim, MD5 je sada zastareo jer je otkriveno da je moguće pronaći dve različite poruke koje generišu isti heš (kolizija).
   * **Gde se koristi**: Nekada je bio široko korišćen za verifikaciju integriteta fajlova, ali se danas izbegava u ozbiljnim kriptografskim primenama zbog svojih slabosti.

**Kako funkcionišu algoritmi za proveru integriteta?**

Svi ovi algoritmi imaju zajednički cilj – generisanje sažetka ili kontrolne vrednosti koja predstavlja originalne podatke. Kada primalac podataka primi podatke, ponovo izračunava ovu vrednost i poredi je sa onom koja je poslana sa podacima. Ako se vrednosti podudaraju, smatra se da podaci nisu promenjeni. Ako postoji razlika, algoritam signalizira da je došlo do greške ili promene u podacima.

Na primer, u sistemima za prenos podataka putem interneta, podaci se dele na male delove ili pakete. Svaki paket može da ima priloženu CRC ili checksum vrednost. Ako paket stigne oštećen, sistem može ponovo zatražiti taj paket, čime se osigurava tačnost podataka.

**Gde se koriste ovi algoritmi?**

Algoritmi za proveru integriteta podataka koriste se u različitim oblastima:

* **Prenos podataka**: U mrežnim protokolima kao što su TCP/IP i Ethernet, CRC se koristi da proveri da li su podaci ispravno preneti.
* **Skladištenje podataka**: Diskovi, memorijske kartice i drugi mediji koriste checksum ili CRC za proveru integriteta podataka.
* **Kriptografija i bezbednost**: SHA algoritmi se koriste u bezbednosnim aplikacijama, kao što su digitalni potpisi, verifikacija identiteta i blockchain.
* **Verifikacija fajlova**: MD5 ili SHA se koriste za proveru autentičnosti preuzetih fajlova, gde korisnik može uporediti heš vrednost dobijenu prilikom preuzimanja sa originalnom vrednošću.

**Zaključak**

Algoritmi za proveru integriteta podataka igraju ključnu ulogu u osiguravanju tačnosti i bezbednosti podataka. Bilo da se koriste za otkrivanje grešaka u prenosu ili za zaštitu osetljivih podataka, ovi algoritmi pomažu da se otkriju promene i potencijalni problemi, obezbeđujući time da podaci ostanu netaknuti i sigurni.

**Checksum**, **CRC**, i **SHA-256** se koriste za proveru integriteta podataka, ali samo **SHA-256** je adekvatan za korišćenje u digitalnim potpisima softvera, kalibracije i konfiguracije. Evo objašnjenja zašto:

**1. Checksum i CRC:**

* **Koriste se za proveru integriteta**, ali nisu dovoljno sigurni za digitalne potpise. Njihova glavna uloga je detekcija slučajnih grešaka u prenosu podataka ili skladištenju. Međutim, oni nisu kriptografski sigurni jer su relativno jednostavni i nisu otporni na maliciozne napade.
* **Ograničenja**:
  + Lako je manipulisati podacima tako da checksum ili CRC vrednost ostane ista, što znači da napadači mogu izmeniti podatke bez otkrivanja.
  + Ne pružaju sigurnost protiv namernih izmena (neautorizovane promene), jer nisu dizajnirani za kriptografske svrhe.

Zbog toga, checksum i CRC se ne koriste za **potpise softvera, kalibracije i konfiguracije**, jer ne garantuju sigurnost i autentičnost.

**2. SHA-256:**

* **SHA-256** je **kriptografski heš algoritam** koji stvara jedinstvenu heš vrednost na osnovu podataka. Za razliku od checksum i CRC, SHA-256 je dizajniran da bude otporan na napade. Kada se koristi u kombinaciji sa asimetričnim ključevima (kao kod digitalnih potpisa), on može osigurati i **integritet** i **autentičnost** podataka.
* **Zašto se koristi za potpise**:
  + Teško je (praktično nemoguće) pronaći dva različita skupa podataka koji generišu isti SHA-256 heš, što osigurava da su podaci autentični.
  + U kombinaciji sa privatnim ključem, SHA-256 omogućava **digitalne potpise**, koji garantuju da je softver, konfiguracija ili kalibracija došla iz pouzdanog izvora i da nije izmenjena.

**SHA-256** se koristi u mnogim sigurnosnim protokolima, kao što su **SSL/TLS** za šifrovanje internetske komunikacije, kao i u **blockchain tehnologiji**, gde je sigurnost ključna. To ga čini pogodnim za digitalne potpise u softveru, kalibraciji i konfiguraciji.

**Zaključak:**

* **Checksum** i **CRC** su korisni za otkrivanje slučajnih grešaka, ali nisu sigurni za potpise.
* **SHA-256** je pogodan za digitalne potpise jer pruža visoku sigurnost, integritet i autentičnost, što ga čini idealnim za zaštitu softvera, kalibracije i konfiguracije.

# Digitalni potpisi (signatures) u embedded sistemima

**Potpisi (signatures)** softvera, kalibracije i konfiguracije su specifične digitalne oznake koje se koriste za proveru autentičnosti, integriteta i verifikaciju tih podataka ili sistema. Oni služe kao potvrda da određeni softver, kalibracija ili konfiguracija potiče od legitimnog izvora i da nije izmenjen od trenutka kada je kreiran. Evo detaljnijeg objašnjenja za svaku od tih oblasti:

**1. Potpisi softvera (Software Signatures)**

**Potpis softvera** je digitalni pečat ili oznaka koju softver dobija kako bi se proverilo da je kreiran od strane pouzdanog izvora i da nije promenjen. Ovo je posebno važno za zaštitu od malicioznih softvera (malware-a) ili zlonamernih izmena. Digitalni potpisi koriste kriptografske tehnike, kao što su **asimetrična enkripcija**, za generisanje jedinstvenog potpisa za određenu verziju softvera.

* **Kako funkcionišu**: Kada softverska kompanija kreira softver, koristi privatni ključ za generisanje potpisa nad datotekama softvera. Kada korisnik preuzima softver, digitalni potpis se proverava koristeći odgovarajući **javni ključ**. Ako se potpis podudara, to znači da softver nije izmenjen i dolazi iz autentičnog izvora.
* **Gde se koristi**: Potpisi softvera se koriste u operativnim sistemima (npr. Windows koristi **Code Signing** za verifikaciju softvera), aplikacijama, drajverima i sigurnosnim sertifikatima. Potpisani softver je prepoznat kao siguran i može se instalirati bez upozorenja.

**2. Potpisi kalibracije (Calibration Signatures)**

**Potpis kalibracije** odnosi se na jedinstvenu oznaku ili identifikaciju koja potvrđuje da su instrumenti ili senzori pravilno kalibrisani prema specifikacijama. Kalibracija je proces podešavanja opreme kako bi se osiguralo da meri ispravno, a potpis kalibracije potvrđuje da je ovaj proces sproveden i da su podaci o kalibraciji tačni.

* **Kako funkcionišu**: Kada se instrument ili senzor kalibriše, generiše se potpis koji je povezan sa specifičnim podešavanjima i standardima koji su korišćeni u kalibraciji. Ovaj potpis često uključuje informacije kao što su datum kalibracije, odgovorni tehničar i upotrebljeni standardi.
* **Gde se koristi**: Potpisi kalibracije su veoma važni u industrijama koje se oslanjaju na tačna merenja, kao što su proizvodnja, automobilska industrija, medicinski uređaji i laboratorijska oprema. Oni osiguravaju da oprema koja se koristi daje tačne i pouzdane rezultate.

**3. Potpisi konfiguracije (Configuration Signatures)**

**Potpis konfiguracije** je oznaka koja potvrđuje da je određena konfiguracija sistema, uređaja ili softverske aplikacije pravilno podešena i da odgovara tačno definisanim standardima. Ova vrsta potpisa se koristi za osiguranje da konfiguracijski fajlovi nisu izmenjeni nakon što su podešeni.

* **Kako funkcionišu**: Kada se sistem ili aplikacija konfiguriše, stvara se digitalni potpis nad konfiguracionim fajlovima koristeći hash algoritme ili kriptografske tehnike. Prilikom kasnijih provera, taj potpis se koristi za verifikaciju da konfiguracija nije neovlašćeno promenjena.
* **Gde se koristi**: Potpisi konfiguracije su ključni u IT infrastrukturi, mrežnim sistemima, serverima i IoT uređajima. Oni osiguravaju da sistem radi pod određenim sigurnosnim i operativnim parametrima i da se ne dogode neovlašćene izmene.

**Zaključak**

**Digitalni potpisi** za softver, kalibraciju i konfiguraciju pružaju siguran i efikasan način da se verifikuje autentičnost, integritet i sigurnost podataka. Oni koriste kriptografske algoritme za osiguranje da su podaci originalni i neizmenjeni, a njihova upotreba je ključna za zaštitu sistema od neovlašćenih modifikacija ili malicioznih napada.

NEW GEN

Digitalni potpisi su kriptografski sistemi koji omogućavaju verifikaciju autentičnosti i integriteta digitalnih podataka. Oni su ključni u mnogim aplikacijama, uključujući sigurnost softvera, e-trgovinu i komunikaciju. Evo kako funkcionišu i kako se primenjuju u embedded programiranju, uz fokus na SHA256 algoritam.

**Digitalni potpisi** su kriptografski alati koji omogućavaju verifikaciju autentičnosti i integriteta digitalnih podataka. Oni funkcionišu slično kao fizički potpisi na papirnim dokumentima, ali su prilagođeni za elektronske dokumente i komunikaciju.

**Osnovna Pojmovi**

1. **Hash Funkcija**: Hash funkcija kao što je SHA256 uzima ulazne podatke (poruku) i pretvara ih u niz fiksne dužine, obično nazvan "hash" ili "digest". SHA256 generiše hash od 256 bita (32 bajta).
2. **Digitalni Potpis**: Digitalni potpis koristi kriptografski privatni ključ da potpiše hash poruke. Ovaj potpis, zajedno sa hash vrednošću i javnim ključem, omogućava drugoj strani da verifikuje integritet i autentičnost poruke.

**Osnovni koncepti digitalnih potpisa**

1. **Kriptografski algoritmi**: Digitalni potpisi koriste javne i privatne ključeve u procesu potpisivanja i verifikacije. Najčešće korišćeni algoritmi za digitalne potpise su RSA, DSA i ECDSA. Algoritmi za hashovanje poput SHA256 su takođe ključni deo ovog procesa.
2. **Proces potpisivanja**:
   * **Hash funkcija**: Prvi korak je primena hash funkcije na podatke koje želimo da potpišemo. Hash funkcija uzima podatke i generiše jedinstveni "otisak" ili "sažetak" koji predstavlja te podatke.
   * **Kriptografsko potpisivanje**: Hash vrednost se zatim potpisuje privatnim ključem potpisa. Rezultat je digitalni potpis koji se može priložiti originalnim podacima.
3. **Proces verifikacije**:
   * **Hash funkcija**: Na strani verifikacije, primatelj koristi istu hash funkciju na primljenim podacima da bi dobio hash vrednost.
   * **Verifikacija potpisa**: Ova hash vrednost se zatim upoređuje sa hash vrednošću koja je dekriptovana iz digitalnog potpisa koristeći javni ključ potpisivača. Ako se podudaraju, podaci su autentični i nisu menjani.

**Proces Digitalnog Potpisivanja**

1. **Generisanje Hash-a**:
   * Originalna poruka (ili dokument) se prolazi kroz SHA256 hash funkciju koja proizvodi jedinstveni hash.
2. **Potpisivanje Hash-a**:
   * Hash vrednost se zatim potpisuje koristeći privatni ključ. Ovaj potpis je jedinstven za određenu poruku i ključ.
3. **Prosljeđivanje Poruke i Potpisa**:
   * Originalna poruka zajedno sa potpisom se šalje krajnjem korisniku.
4. **Verifikacija**:
   * Primalac koristi javni ključ da verifikuje potpis. On prvo izračunava hash poruke koju je primio i upoređuje ga sa hash vrednošću dobijenom iz potpisa.

**Zašto je SHA256 Pogodan za Digitalne Potpise**

1. **Bezbednost**:
   * SHA256 je deo SHA-2 porodice hash funkcija, koja je dizajnirana da bude otporna na kolizije (situacije kada različiti ulazi proizvode isti hash) i da bude kriptografski sigurna. Ovo je ključno za digitalne potpise jer omogućava da se menja samo potpis i ne i sadržaj poruke.
2. **Fiksna Dužina Hash-a**:
   * SHA256 proizvodi hash od 256 bita, što je dovoljno dugačko da obezbedi robustan nivo zaštite protiv napada.
3. **Brzina i Efikasnost**:
   * SHA256 je dovoljno brz da se koristi u mnogim real-time i embedded sistemima, a pritom nudi visoku sigurnost.

**SHA256 i digitalni potpisi**

**SHA256** (Secure Hash Algorithm 256-bit) je popularan algoritam za hashovanje koji se koristi u digitalnim potpisima zbog svojih karakteristika:

1. **Sigurnost**: SHA256 proizvodi 256-bitne hash vrednosti, što pruža visok nivo sigurnosti. Ovo je dovoljno dug za većinu primena, što otežava pokušaje napada poput kolizija (kada dva različita skupa podataka proizvode isti hash).
2. **Jedinstvenost**: SHA256 osigurava da svaki jedinstveni ulaz proizvodi jedinstveni hash. Ako se podaci promene, hash će se značajno promeniti, što pomaže u očuvanju integriteta podataka.
3. **Brzina**: Iako SHA256 može biti sporiji od nekih drugih hash funkcija, njegova brzina je i dalje dovoljna za mnoge aplikacije, uključujući i embedded sisteme gde su resursi često ograničeni.

**Primena u Embedded Programiranju**

U embedded sistemima, digitalni potpisi se koriste za zaštitu firmware-a, konfiguracijskih datoteka, i komunikacijskih protokola. Evo kako se može primeniti:

1. **Potpisivanje Firmware-a**:
   * Kada se firmware razvija, hash njegovog sadržaja se izračunava koristeći SHA256, a zatim se taj hash potpisuje privatnim ključem. Kada se firmware instalira na uređaj, uređaj koristi javni ključ za verifikaciju potpisa, osiguravajući da firmware nije modifikovan.
2. **Bezbedna Komunikacija**:
   * U uređajima sa ograničenim resursima, digitalni potpisi se koriste za autentifikaciju poruka i komandi između uređaja i servera, obezbeđujući da poruke nisu promenjene i da dolaze od pouzdane strane.
3. **Održavanje Integriteta Podataka**:
   * Digitalni potpisi pomažu u očuvanju integriteta podataka u aplikacijama gde je važno da podaci ne budu izmenjeni tokom prenosa ili skladištenja.

Ovaj pristup pomaže da se obezbedi sigurnost u okruženju sa ograničenim resursima, što je često slučaj u embedded sistemima.

**Primena u embedded programiranju**

U embedded programiranju, digitalni potpisi se koriste za:

1. **Verifikaciju firmware-a**: Pre nego što se firmware učita na uređaj, može se proveriti digitalni potpis kako bi se osiguralo da firmware nije modifikovan i da dolazi iz pouzdane izvore.
2. **Autentifikaciju komunikacije**: Digitalni potpisi mogu se koristiti za osiguranje da su poruke koje uređaji razmenjuju zaista poslate od strane ovlašćenog izvora i da nisu modifikovane tokom prenosa.
3. **Zaštitu od neovlašćenog pristupa**: Uređaji mogu koristiti digitalne potpise za autentifikaciju korisnika ili drugih uređaja, čime se sprečava neovlašćeni pristup.

**Zaključak**

SHA256 je pogodna za digitalne potpise zbog svoje visoke sigurnosti i sposobnosti da proizvede jedinstvene hash vrednosti. U embedded sistemima, ova tehnologija pomaže u očuvanju integriteta i autentičnosti podataka i firmware-a, što je ključno za sigurnost i pouzdanost uređaja.