Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621
DOI: https://doi.org/10.24867/11BE10Nikic

SIGURNA KLIJENT-SERVER KOMUNIKACIJA KORIŠĆENJEM KRIPTOČIPA I NB-IOT KOMUNIKACIONOG MODULA

SECURE CLIENT-SERVER COMMUNICATION USING CRYPTOCHIP AND NB-IOT COMMUNICATION MODULE

Dušan Bortnik, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu je prikazana specifikacija i implementacija sigurne klijent-server komunikacije uz upotrebu kriptočipa putem NB-IoT mreže. Sigurna klijent-server veza je hardverski realizovana uz pomoć ATECC608a kriptočipa. NB-IoT komunikacija je ostvarena pomoću Quectel BC68 modula. Programska podrška je realizovana u programskom jeziku C na mikrokontroleru ATSAML21J18B. Implementiran je i testiran bazični TLS1.3 protokol stek.*

Ključne reči: *TLS1.3, ATECC608a kriptočip, BC68 NB-IoT modul, TCP, HTTPs*

Abstract – *This paper examines specification and implementation of secure client-server communication with the usage of cryptochip over NB-IoT network. Secure client-server connection is hardware-based, established using ATECC608a cryptochip. NB-IoT communication is achieved using Quectel BC68 module. Software, written in C, is executed on microcontroller ATSAML21J18B. Basic TLS1.3 protocol is implemented and tested.*

Keywords: *TLS1.3, ATECC608a cryptochip, BC68 NB-IoT module, TCP, HTTPs*

1. UVOD

Na polju mrežnih komunikacija, uvek je aktuelno pitanje sigurnosti. Pod sigurnom komunikacijom podrazumeva se komunikacija koja obuhvata tri stvari. To su poverljivost informacija (eng. *confidentiality*), očuvanost poruke (eng. *message integrity*) i identifikacija krajnjih učesnika komunikacije (eng. *end-point authentication*).

Pod poverljivošću informacija podrazumeva se da samo učesnici u komunikaciji (pošiljalac i primalac poruke) budu u stanju da razumeju sadržaj same poruke. Kako bi ovo bilo postignuto koriste se različite tehnike enkripcije.

Očuvanost poruke podrazumeva da sadržaj poruke ostaje netaknut u toku njenog slanja.

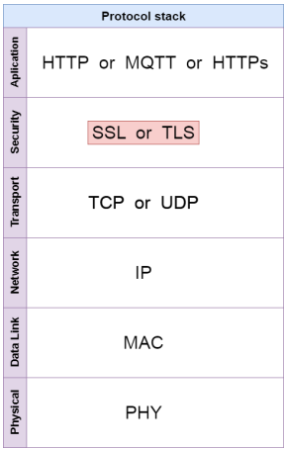
Različite *checksum* tehnike je moguće primeniti kako bi ova osobina bila zadovoljena. Konačno, kod identifikovanja krajnjih učesnika komunikacije i pošiljalac i primalac su u stanju da uspešno prepoznaju suprotnu stranu u komunikaciji, odnosno da dokažu da je suprotna strana baš ona kojom se predstavlja.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Ivan Mezei, vanr. prof.

Ovo se rešava upotrebom različitih autentikacionih protokola. Više o ovome moguće je naći u literaturi [1].

Najznačajniji i najkorišćeniji protokol dizajniran da obezbedi sigurnu mrežnu komunikaciju jeste TLS protokol.

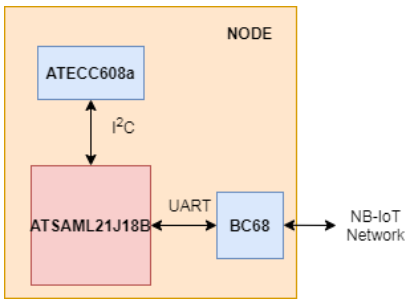
Transport Layer Security (skraćeno TLS) predstavlja dodatni sloj u arhitekturi mreže (eng. *protocol stack*), čija uloga jeste ostvarivanje sigurnog kanala između prijemne i predajne strane. *Protocol stack* je prikazan na slici 1. TLS zapravo predstavlja poboljšanu TCP konekciju, odnosno TCP konekciju u kojoj se primenjuju različite kriptografske tehnike kako bi se obezbedile sve navedene osobine sigurne komunikacije – poverljivost informacija, očuvanost poruke i identifikacija krajnjih učesnika komunikacije. Prema tome, TLS je zapravo „siguran" TCP protokol, pa se stoga može koristiti svugde gde postoji gotova TCP infrastruktura. U okviru ovog rada realizovana su poslednja tri sloja *protocol stack-a* (transportni, bezbednosni i aplikacioni sloj) sa posebnim naglaskom na bezbednosni. U drugom poglavlju diskutovane su korišćene tehnologije, dok je u poglavlju tri dat osvrt na sam koncept rešenja i realizaciju.



Slika 1. *Protocol stack*

2. KORIŠĆENE TEHNOLOGIJE

Kompletna klijent-server komunikacija između jednog čvora u mreži i udaljenog servera realizovana je korišćenjem mikrokontrolera ATSAML21J18B i dodatnih periferijskih komponenti – kriptočipa ATECC608a i NB-IoT modula BC68. Sve korišćene komponente su integrisane na PCB-u koji predstavlja jedan čvor (eng. *node*) u mreži. Pojednostavljena struktura čvora prikazana je na slici 2.



Slika 2. *Struktura čvora*

2.1. ATSAML21J18B mikrokontroler

Microchip-ov 32-bitni mikrokontroler izuzetno niske potrošnje ATSAML21J18B obezbeđuje funkcionalnost čvora u mreži. Pored niske potrošnje energije, odlikuje ga i ARM Cortex-M0+ mikroprocesor, 256kB Flash memorije i temperaturni opseg od -40°C do 105°C.

2.2. NB-IoT komunikacija

Veza sa serverom putem *Narrowband IoT* (NB-IoT) mreže je ostvarena pomoću modula BC68. Komunikacija mikrokontrolera i BC68 modula se vrši putem UART-a.

Quectel BC68 predstavlja NB-IoT modul koji omogućava rad sa različitim mrežnim protokolima uz nisku potrošnju energije. Iz tog razloga često nalazi primenu u različitim IoT aplikacijama. Za komunikaciju sa modulom koriste se standardne AT komande (*test*, *read*, *write* i *execution*) date u sledećem formatu:

$$AT+<cmd>x \qquad (1)$$

gde x predstavlja vrednosti $\{=?,\ \ ,=<\cdots>,?\}$ u zavisnosti od tipa komande.

Nakon procesiranja svake poruke modul vraća odgovarajuću povratnu informaciju i nakon toga, indikaciju uspešnosti operacije (OK, ERROR ili +CME ERROR:<err>).

Pored standardnih komandi za inicijalizaciju modula, za realizaciju TCP sloja od interesa su sledeće komande: AT+NSOCR, AT+NSOCO, AT+NSOSD, AT+NSORF i AT+NSOCL.

AT+NSOCR komanda obavlja kreiranje soketa i njegovu konfiguraciju. Kao povratna informacija dobija se broj *socket*-a i indikacija uspešnosti OK (ili kod greške, ukoliko greška postoji).

AT+NSOCO komanda vrši uspostavljanje peer-to-peer konekcije sa udaljenim serverom na odgovarajućem *socket*-u. Kao povratna informacija dobija se indikacija uspešnosti OK (ili kod greške, ukoliko greška postoji).

AT+NSOSD komanda vrši slanje TCP paketa ka odgovarajućem serveru. Kao povratna informacija dobija se broj *socket*-a i broj bajtova poslate informacije (format poruke <soc>,<len>), kao i indikacija uspešnosti OK (ili kod greške, ukoliko greška postoji).

AT+NSORF komanda vrši prijem podataka na odgovarajućem *socket*-u u traženoj količini. Kada podaci pristignu na *socket* generisaće se +NSONMI kao indikacija pristiglih podataka. Kao povratna informacija dobija se broj *socket*-a, IP adresa servera, port, broj bajtova primljene informacije, podaci i preostala dužina podataka koju je potrebno pročitati (format poruke <soc>,<ip_addr>,<port>,<len>,<data>,<rem_len>), kao i indikacija uspešnosti OK (ili kod greške, ukoliko greška postoji).

AT+NSOCL komanda obavlja zatvaranje odgovarajućeg *socket*-a. Kao povratna informacija dobija se indikacija uspešnosti OK (ili kod greške, ukoliko greška postoji).

Više informacija o ovom modulu i ostalim komandama moguće je naći u *datasheet*-u, datom u [2].

2.3. Kriptočip ATECC608a

Sve kriptografske operacije neophodne za realizaciju TLS sloja implementirane su korišćenjem *Microchip*-ovog kriptočipa *ATECC608a*.

ATECC608a kriptočip je integrisano kolo koje omogućava čuvanje kriptografskih ključeva, kao i akceleraciju različitih kriptografskih algoritama. Čip sadrži EEPROM memoriju u kojoj je moguće čuvati do 16 različitih privatnih ključeva, sertifikata i ostalih „tajnih“ podataka. Pored toga, postoji i statička RAM (SRAM) memorija u kojoj je moguće čuvanje *privremenih* ključeva i različitih međurezultata koji su validni sve dok postoji napon napajanja.

Komunikacija sa kriptočipom se vrši putem standardnog I^2C interfejsa. Kriptočip podržava generisanje parova javni-privatni ključ korišćenjem *NIST standard P256 prime* krive uz ECDH (*Elliptic-curve Diffie–Hellman*) *key-agreement* i ECDSA *signature verification* protokol. Takođe, podržani su i standardni algoritmi kao što su AES-128, SHA256 i različite varijacije SHA (HMAC, PRF i HKDF). Pored toga, u okviru kriptočipa nalazi se i generator pseudo-slučajnih brojeva visoke entropije koji, između ostalog, omogućava i generisanje privatnih ključeva.

Prilikom realizacije TLS sloja, od interesa su sledeće komande u okviru kriptočipa: AES, ECDH, GenKey, KDF, Nonce i SHA.

AES komanda izvršava standardni AES algoritam u ECB ili GCM modu (enkripciju i dekripciju).

ECDH komanda generiše *pre-master secret* na osnovu klijentovog privatnog i serverovog javnog ključa ili obrnuto.

GenKey komanda služi za generisanje ECC javnog ključa na osnovu odgovarajućeg privatnog ključa.

KDF komanda implementira odgovarajuću *key derivation* funkciju. Od posebnog značaja za TLS protokol je **HKDF** funkcija (postoji kao ugrađeni mod KDF komande).

Nonce komanda omogućava upis vrednosti na ulazu u SRAM registar (TempKey).

SHA komanda izračunava SHA-256 funkciju koja na izlazu kao rezultat daje obrađeni 32-bitni heš.

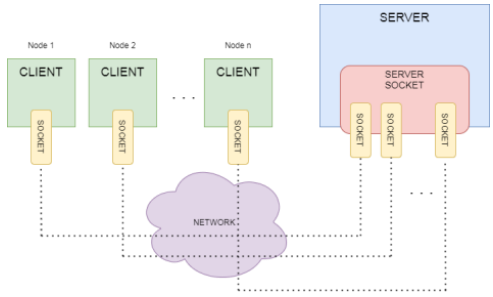
Glavni razlog za upotrebu kriptočipa ogleda se u značajno većoj brzini izvršavanja kriptografskih operacija u hardveru u odnosu na standardna softverska rešenja. Takođe, zahvaljujući internoj memoriji kriptočipa u okviru koje je moguće smeštanje ključeva, verovatnoća njihovog otkrivanja je svedena na minimum.

Više informacija o kriptočipu ATECC608a i ostalim komandama moguće je naći u [3].

2.3. Klijent-server arhitektura

Čitav sistem sa n čvorova i jednim serverom implementira standardnu centralizovanu klijent-server arhitekturu. Klijent-server arhitektura prikazana je na slici 3.

Svaki od *n* čvorova šalje zahtev radi ostvarivanja komunikacije sa serverom putem radio komunikacionog kanala (NB-IoT mreža). Server predstavlja udaljeni računar koji sakuplja informacije sa pojedinačnih čvorova, obrađuje ih na odgovarajući način i šalje povratnu informaciju. Način same komunikacije opisan je u nastavku.



Slika 3. *Klijent-server arhitektura*

3. KONCEPT REŠENJA I REALIZACIJA

U naredna tri poglavlja opisan je koncept rešenja i implementacija poslednja tri sloja *protocol stack*-a.

3.1. Realizacija transportnog sloja – TCP

Transmission Control Protocol (skraćeno TCP) protokol razvijen je upotrebom gore navedenih komandi NB–IoT modula BC68.

Na samom početku, vrši se kreiranje *socket*-a na odgovarajućoj IP adresi i odgovarajućem portu. Nakon toga, potrebno je uspostaviti TCP konekciju sa serverom. Nakon uspešnog uspostavljanja konekcije, moguće je vršiti slanje ka i primanje podataka od strane servera. Prilikom okončavanja komunikacije sa serverom, potrebno je zatvoriti prethodno otvoreni *socket*.

Primer TCP komunikacije sa serverom koja se sastoji od slanja i primanja jedne poruke korišćenjem BC68 modula prikazana je na sledećem listingu.

```
//Create a socket using NSOCR command
AT+NSOCR=STREAM,6,0,1
// Received socket number
1
// Success flag
OK
//Connect to the server using NSOCO command
AT+NSOCO=1,192.247.24.38,1234
// Success flag
OK
//Send message using NSOSD command
AT+NSOSD=1,4,01020304,0x100,101
// Received: socket number, number of data being sent
1,4
// Success flag
OK
//NSONMI unsolicited message - data has been received
+NSONMI:1,4
//Read received data using NSORF command
AT+NSORF=1,4
// Received: socket number, IP address, port number, number of
data being received, data, remaining data
1,192.247.24.38,1234,4,1a2b3c4e,0
// Success flag
OK
//Close socket using NSOCL command
AT+NSOCL=1
// Success flag
OK
```

Listing 1. *TCP komunikacija pomoću BC68 modula*

3.2. Realizacija bezbednosnog sloja – TLS

Nakon realizacije TCP komunikacije, moguće je implementirati TLS protokol kao njegovo unapređenje. U okviru ovog rada implementirana je poslednja verzija TLS protokola 1.3. TLS1.3 *handshake* protokol se sastoji iz nekoliko osnovnih faza. Ove faze, kao i čitav *handshake* deo protokola prikazan je na slici 4. Inicijalno, nakon uspostavljanja TCP konekcije sa serverom, klijent šalje *Client Hello* poruku u okviru koje je se nalazi lista *cipher suite*-ova koje klijent podržava, kao i klijentov javni ključ. Uzimajući u obzir da je pomoću datog kriptočipa moguće izvršavati AES128 GCM enkripciju/dekripciju i SHA256 heš algoritam, jedini validan *cipher suite* koji klijent podržava i prosleđuje serveru je TLS_AES_128_GCM_SHA256 *cipher suite*. Klijentov javni ECC ključ se generiše korišćenjem genKey komande. Nakon toga, server šalje *Server Hello* paket u okviru kog se nalazi serverov javni ključ i izabrani *cipher suite* (u ovom slučaju jedini koji klijent podržava – TLS_AES_128_GCM_SHA256). Nakon *Server Hello* paketa, server šalje *Server Encrypted Extensions, Server Certificate, Server Certificate Verify* i *Server Finished* pakete. Svi paketi su enkriptovani pomoću *server handshake key*-a i na kraju uvek poseduju tag za autentikaciju (*Auth tag*). U međuvremenu, klijent je dužan da pomoću svog privatnog ključa i serverovog javnog ključa odredi *shared secret* pomoću ECDH komande kriptočipa. Takođe, potrebno je izvršiti i izračunavanje heša svih *handshake* poruka do tog trenutka pomoću SHA komande kriptočipa. Nakon toga, vrši se računanje četiri *handshake* ključa: *client handshake* ključa, *server handshake* ključa, *client handshake* inicijalizacionog vektora (IV) i *server handshake* inicijalizacionog vektora (IV). Ovo se radi korišćenjem *HKDF-Extract* i *HKDF-Expand* funkcija. Ove funkcije moguće je implementirati pomoću KDF komande kriptočipa. Uzimajući u obzir da KDF komanda obavlja jednu iteraciju HMAC-SHA256 algoritma, ove funkcije su implementirane na osnovu njihove specifikacije date u [4]. Pseudokod koji demonstrira način na koji se formiraju klijentski ključevi dat je na sledećem listingu.

```
early_secret = HKDF-Extract(
  salt = 00, key = 0x00000000)
empty_hash = SHA256(empty string)
derived_secret = HKDF-Expand(
  key = early_secret,
  label = "derived", context = empty_hash, output_length = 32)
handshake_secret = HKDF-Extract(
  salt = derived_secret, key = shared_secret)
client_handshake_traffic_secret = HKDF-Expand(
  key = handshake_secret,
  label = "c hs traffic", context = hello_hash, output_length = 32)
client_handshake_key = HKDF-Expand(
  key = client_handshake_traffic_secret,
  label = "key", context = empty string, output_length = 16)
client_handshake_iv = HKDF-Expand(
  key = client_handshake_traffic_secret,
  label = "iv", context = empty string, output_length = 12)
```

Listing 2. *Pseudokod za računanje handshake ključeva*

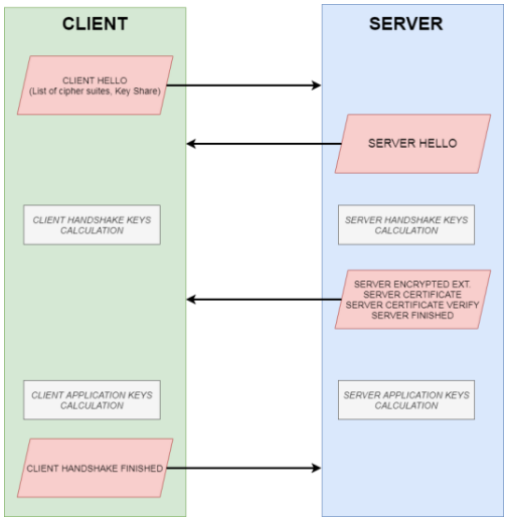
Serverski ključevi se računaju na isti način kao i klijentski, uzimajući u obzir jednu izmenu, ime labele. U slučaju servera, prilikom računanja *server handshake traffic secret*-a koristi se labela sa nazivom „*s hs traffic*”.

Pomoću ovih ključeva je sada moguće kompletirati čitav *handshake* protokol. Kao prvo, potrebno je izvršiti dekripciju četiri poruke ranije pristigle od strane servera pomoću *server handshake* ključa i IV-a. Nakon njihove dekripcije, klijent je u obavezi da izračuna *verify data* i pošalje finalni paket ka serveru i time kompletira *handshake* deo protokola. *Finished key* se računa na osnovu *client traffic handshake secret*-a pomoću KDF komande kriptočipa. Pored toga, pomoću SHA komande potrebno je izračunati i SHA256 heš svih primljenih poruka (dekriptovanih) do tog trenutka. Konačno, na osnovu *finished key*-a i izračunatog heša moguće je izračunati *verify data*. Tako izračunat, enkriptuje se pomoću *client handshake* ključa i IV-a, a potom se prosleđuje ka serveru. Time je *handshake* deo protokola završen. Pseudokod ovih operacija dat je na listingu 3.

```
finished_key = HKDF-Expand(
   key = client_handshake_traffic_secret,
   label = "finished", context = empty string, output_length = 32)
finished_hash = SHA256(all messages from Client Hello to Server Finished)
verify_data = HMAC-Extract(
   salt = finished_key,
   key = finished_hash)
```

Listing 3. *Pseudokod za računanje verify data*

Na vrlo sličan način, implementira se i računanje ključeva za primopredaju aplikacionog sadržaja. Nakon računanja aplikacionih ključeva i aplikacionih inicijalizacionih vektora moguće je vršiti razmenu podataka između dve strane. Klijent koristi isti enkripcioni algoritam i uz upotrebu izračunatih aplikacionih ključeva enkriptuje korisne podatke, određuje autentikacioni tag i šalje poruku ka serveru. Shodno tome, server šalje odgovarajuću povratnu informaciju.



Slika 4. *TLS1.3 handshake deo protokola*

3.3. Realizacija aplikacionog sloja – HTTP i HTTPs

Hypertext Transfer Protocol (skraćeno HTTP) je protokol aplikacionog sloja koji se, u najopštijem smislu, bazira na slanju komandi ka serveru (*request*) i prijemu povratne informacije od strane servera (*response*). Naime, nakon uspostavljanja TCP konekcije između klijentske i serverske strane, klijent šalje neku od postojećih HTTP komandi (GET, HEAD, PUT, OPTIONS itd.) ka serveru u odgovarajućem formatu. U zavisnosti od poslate komande, server šalje adekvatan odgovor.

Ukoliko između transportnog i aplikacionog sloja postoji i bezbednosni sloj, odnosno ukoliko se umesto TCP konekcije implementira TLS konekcija, ostvaruje se *Hypertext Transfer Protocol Secure* (skraćeno HTTPs) protokol. Na ovaj način se obezbeđuje sigurni kanal ka serveru preko nezaštićene mreže. Postojeće komande HTTP protokola su i dalje validne, s tim da se u slučaju HTTPs protokola one šalju ka serveru tek nakon što se ostvari TLS *handshake* faza između klijenta i servera. Pored toga, one su enkriptovane klijentskim aplikacionim ključem i IV-om izračunatim nakon *handshake*-a. Drugim rečima, u HTTPs protokolu, HTTP komande se šalju kao aplikacioni sadržaj u okviru TLS protokola. Time je HTTPs protokol implementiran.

4. ZAKLJUČAK

U okviru ovog rada implementirana su poslednja tri sloja internet *protocol stack*-a, sa naglaskom na sigurnosni sloj. Sve kriptografske operacije u okviru TLS protokola izvršene su pomoću kriptočipa ATECC608a. Glavni razlog za njegovu upotrebu bila je značajno veća brzina izvršavanja operacija hardverskim putem, u odnosu na iste operacije implementirane pomoću softvera. Komunikacija sa serverom putem NB-IoT mreže ostvarena je pomoću *Quectel*-ovog modula BC68. Celokupan kod izvršava se na mikrokontroleru ATSAML21J18B i napisan je u programskom jeziku *C*.

Prema tome, omogućena je sigurna komunikacija između čvorova na *edge*-u (klijent) i centralizovane jedinice (server) putem nebezbedne mreže, što je i bio cilj.

Budući da je u ovom radu implementiran bazični TLS protokol u okviru koga server ne zahteva sertifikat od strane klijenta, dodatni prostor za napredak u pogledu bezbednosti ogleda se u mogućnosti implementacije takve komunikacije, u kojoj bi se od klijenta tražilo da serveru dostavi svoj sertifikat. Na taj način, sigurnost bi bila dodatno poboljšana. Takođe, ukoliko bi klijent iskoristio *session ticket*-e poslate od strane servera, značajan napredak bio bi ostvaren i u pogledu brzine uspostavljanja nove TLS konekcije, odnosno njenog nastavka (*resumption*). Međutim, nedostatak ovoga bila bi donekle smanjena sigurnost komunikacije.

5. LITERATURA

[1] J. Kurose, K. Ross, "Computer Networking: A Top-Down Approach 6th Edition", *Pearson*, pp. 672-755, 2013.

[2] Quectel BC68 Datasheet: https://www.quectel.com/product/bc68.html (pristupljeno u septembru 2020.)

[3] Microchip ATECC608a datasheet: https://www.microchip.com/wwwproducts/en/ATECC608A (pristupljeno u septembru 2020.)

[4] RFC8446 : https://tools.ietf.org/html/rfc8446 (pristupljeno u septembru 2020.)

Kratka biografija:

Dušan Bortnik rođen je u Novom Sadu 1996. god. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Embedded sistemi i algoritmi odbranio je 2019.god.
kontakt: *bortnik@uns.ac.rs*