ECC za digitalno potpisivanje diploma i dokumenata na Sveučilištu u Mostaru

Robert Rozić

10. prosinca 2024.

1 Uvod

Problem s kojim se akademska zajednica susreće je autentičnost i sigurnost akademskih dokumenata koji se distribuiraju putem digitalnih kanala.

IT centar Sveučilišta u Mostaru (SUMIT)[1] održava Informacijski sustav Sveučilišta (ISS), a jedna od ključnih funkcionalnosti je generiranje dokumenata poput diploma, potvrda o statusu studenta i prijepisa ocjena. SUMIT također održava sustav eMatica, iz kojeg se generiraju svjedodžbe o završenim razredima, potvrde o maturi te sustav eUpisi, koji upravlja upisima studenata i učenika.

Korištenjem kriptografije zasnovane na eliptičnim krivuljama (ECC) npr. ECDSA (Elliptic Curve Digital Signature Algorithm) može se implementirati digitalno potpisivanje ovih dokumenata, osiguravajući njihovu autentičnost i integritet. Svaka osoba zadužena za potpisivanje dokumenata dobila bi potpisni certifikat, čime bi se omogućila jedinstvena i sigurnosno zaštićena identifikacija.

SUMIT bi izradio aplikaciju za potpisivanje te izdavanje i provjeru vjerodostojnosti certifikata, slično sustavu Certilia koji se koristi u Republici Hrvatskoj. Sveučilište u Mostaru je u kontaktu i u suradnji s AKD-om, koji je razvio Certilia aplikaciju. Ova integracija osigurava sigurnost, pouzdanost i jednostavnost digitalnog potpisivanja i provjere.

Primjer primjene je izrada jednostavne koncept aplikacije za digitalni potpis koja je opisana u nastavku ovog dokumenta. Za izradu se koristila openssl biblioteka na operacijskom sustavu Linux (distribucija Fedora), programski jezik Python te Python biblioteke ECDSA i pyhanko.

Implementacija i programski kod javno su dostupni na GitHub platformi.

2 Kriptografija eliptičnih krivulja (ECC)

Kriptografija eliptičnih krivulja (Eliptic Curve Cryptography - ECC) je pristup kriptografiji s javnim ključem koji se temelji na algebarskoj strukturi eliptičkih krivulja nad konačnim poljima. Jedna od glavnih prednosti u usporedbi s non-ECC kriptografijom je ista razina sigurnosti koju pružaju ključevi manje veličine.[2]

Korištenje eliptičnih krivulja u kriptografiji neovisno su predložili Neal Koblitz[3] i Victor S. Miller[4] 1985. Algoritmi kriptografije eliptičkih krivulja ušli su u široku upotrebu od 2004. do 2005.

Na konferenciji RSA 2005., Agencija za nacionalnu sigurnost (NSA) najavila je Suite B, koji isključivo koristi ECC za generiranje digitalnog potpisa i razmjenu ključeva. Paket je namijenjen za zaštitu klasificiranih i neklasificiranih nacionalnih sigurnosnih sustava i informacija.[5]

Nacionalni institut za standarde i tehnologiju (NIST) odobrio je kriptografiju eliptičke krivulje u svom skupu preporučenih algoritama Suite B, posebno Diffie-Hellman eliptičke krivulje (ECDH) za razmjenu ključeva i Algoritam digitalnog potpisa eliptičke krivulje (ECDSA) za digitalni potpis. NSA dopušta njihovu upotrebu za zaštitu podataka klasificiranih do strogo povjerljivih s 384-bitnim ključevima.[6]

NIST(National Institute of Standards and Technology) preporuka FIPS (Federal Information Processing Standards) 186-4 iz 2013. sadrži ukupno pet prostih krivulja i deset binarnih krivulja. Krivulje su odabrane za optimalnu sigurnost i učinkovitost implementacije.[7]
FIPS 186-5 je 3. veljače 2023. zamijenio FIPS 186-4.[8] Neke od glavnih izmjena standarda uključuju (Slika 1):

- Uklanjanje DSA (Digital Signature Algorithm) koji više nije odobren za generiranje digitalnih potpisa u FIPS 186-5. Može se koristiti samo za verifikaciju potpisa generiranih prema FIPS 186-4.
- Uveden je Edwards-Curve Digital Signature Algorithm (EdDSA) koji je razvijenija verzija algoritma baziranog na eliptičnim krivuljama koji koristi ključeve još manje veličine.
- Što se tiče samih preporučenih eliptičnih krivulja, eliptične krivulje nad binarnim poljima, koje su bile specificirane u FIPS 186-4, sada su zastarjele. Preporučene su krivulje poput Montgomery i Edwards krivulja, kao što su Curve25519 i Edwards448, umjesto starijih binarnih krivulja.
- Krivulja secp256k1, koja se često koristi u blockchain aplikacijama, spominje se i dozvoljena je za korištenje.

Table 2. Allowed Usage of the Specified Curves

Specified Curves	Allowed Usage
K-233, B-233 K-283, B-283 K-409, B-409 K-571, B-571	Deprecated
P-224 P-256 P-384 P-521 Edwards25519 Edwards448	ECDSA, EC key establishment (see [SP_800-56A]) EdDSA
Curve25519, W-25519 Curve448, E448, W-448	Alternative representations included for implementation flexibility. Not to be used for ECDSA or EdDSA directly.

Slika 1: FISP Preporučene krivulje

2.1 Prednosti ECC

Brzo generiranje ključa Stvaranje ECC ključa jednostavno je poput sigurnog stvaranja nasumičnog cijelog broja u određenom rasponu, što ga čini vrlo brzim. Bilo koji cijeli broj u rasponu predstavlja važeći ECC tajni ključ. Javni ključevi u ECC su EC točke, koje su parovi cjelobrojnih koordinata x i y koje leže na krivulji. To je značajno u aplikacijama koje zahtijevaju brzu obradu podataka.

Manja potrošnja resursa Zahvaljujući manjim ključevima, ECC koristi manje procesorske snage i memorije što je vrlo korisno u uređajima s ograničenim resursima, poput pametnih kartica, mobilnih uređaja ili IoT uređaja.

Visoka sigurnost 256-bitni ECC javni ključ osigurava usporedivu sigurnost s 3072-bitnim RSA javnim ključem. Uz ECC, možete dobiti istu razinu sigurnosti s manjim ključevima.[9]

2.2 Nedostaci ECC

Veličina enkripcije ECC povećava veličinu šifrirane poruke znatno više od RSA enkripcije.

Kompleksna sigurnost Ako je veličina korištenog ključa dovoljno velika, ECC se smatra vrlo sigurnim. Za internu komunikaciju, američka vlada treba ECC s veličinom ključa od 256 ili 384 bita, ovisno o razini osjetljivosti materijala.

Podrška Iako je ECC postao popularan, još uvijek može postojati manja podrška za ECC u određenim legacy sustavima ili aplikacijama u usporedbi s algoritmima poput RSA.

Složenost implementacije ECC algoritam je teži za implementaciju od RSA.

2.3 Primjene ECC

- Digitalno potpisivanje
- Razmjena ključeva
- Enkripcija podataka
- Blockchain i kriptovalute
- Identifikacija i autentifikacija

3 Implementacija

U našem primjeru ECC ćemo koristiti za izradu koncepta aplikacije za digitalni potpis.

3.1 Razrada aplikacije

1. Kreiranje vlastitog certifikacijskog autoriteta (CA)
2. Generiranje potpisnog certifikata korisnika
3. Digitalno potpisivanje dokumenta
4. Provjera valjanosti potpisa

3.2 Kreiranje vlastitog CA

Za potrebe testa kreirat ćemo vlastiti CA (Certificate Authority) koji će predstavljati Sveučilište u Mostaru, a taj CA će izdavati i potpisivati certifikate potrebne za digitalni potpis.

Koristit ćemo EC ključ generiran putem linux terminala koji koristi NIST preporučenu P384 eliptičnu krivulju.

```
openssl ecparam -genkey -name secp384r1 -out ca_private_key.pem
```

Zatim generiramo self-signed certifikat za CA. Ovaj certifikat će koristiti korisnici i uređaji kako bi vjerovali CA SUM i certifikatima koje izdaje. Unutar openssl_ca.cnf smo definirali podatke o CA za Sveučilište u Mostaru.

```
openssl req -x509 -new -nodes -key ca_private_key.pem -sha256 -days 3650
-out ca_cert.pem -config ca_openssl.cnf
```

Trenutno posjedujemo privatni ključ CA za potpisivanje certifikata koji će služiti za digitalne potpise na razini organizacijske jedinice SUM.

3.3 Generiranje potpisnog certifikata korisnika

Krenut ćemo s jednostavnim generiranjem ključeva i spremanjem u .pem obliku, što je standard za spremanje kriptografskih ključeva.

Za generiranje ključeva ovaj put ćemo koristiti Python skriptu i biblioteku ecdsa.

3.3.1 Biblioteka ECDSA

Ova biblioteka nudi jednostavno rukovanje s ECC, uključujući podršku za ECDSA (Elliptic Curve Digital Signature Algorithm), EdDSA (Edwards Curve Digital Signature Algorithm) i ECDH (Elliptic Curve Diffie-Hellman).[10]

Instalaciju vršimo pozivanjem sljedeće naredbe u terminalu:

```
pip install ecdsa
```

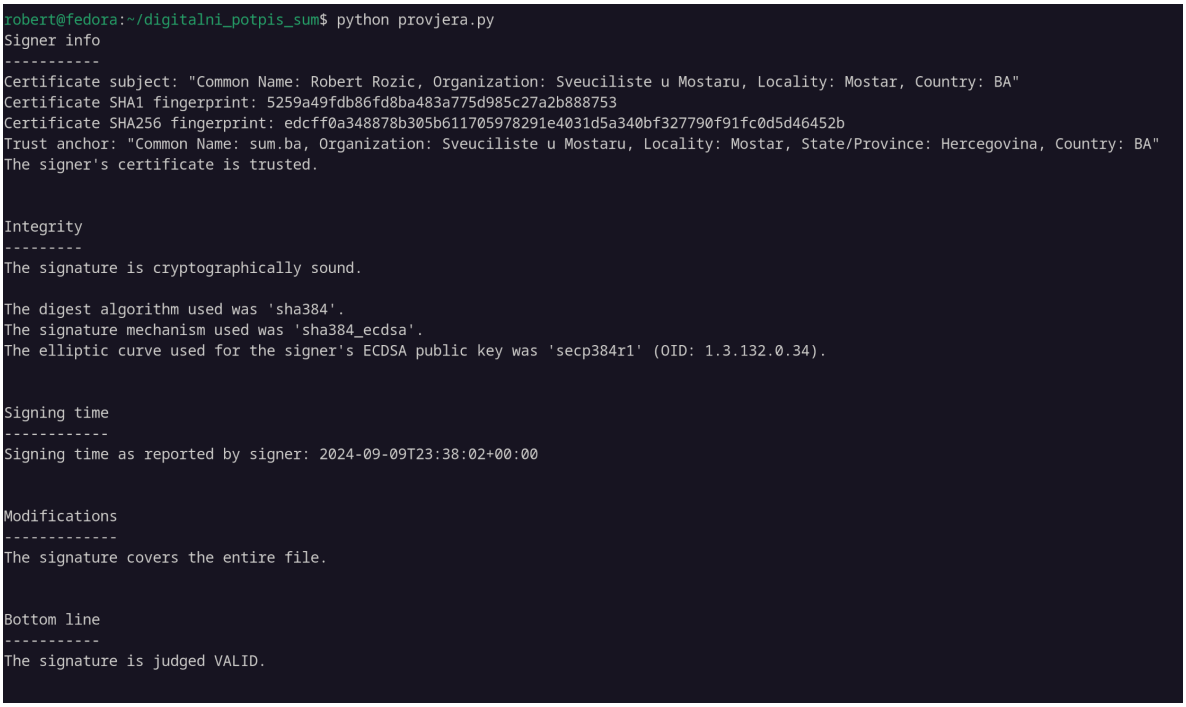
Za generiranje ključeva ponovno koristimo P384 krivulju.

```python
# Import biblioteke i elipticne krivulje
from ecdsa import SigningKey, NIST384p

# Generiranje privatnog kljuca s P-384 krivuljom
private_key = SigningKey.generate(curve=NIST384p)
print("Privatni kljuc:", private_key)

# Generiranje javnog kljuca iz privatnog
public_key = private_key.get_verifying_key()
print("Javni kljuc:", public_key)

# Spremanje privatnog i javnog kljuca u .pem formatu
with open("private_key.pem", "wb") as priv_file:
    priv_file.write(private_key.to_pem())

with open("public_key.pem", "wb") as pub_file:
    pub_file.write(public_key.to_pem())
```

Kao rezultat dobili smo privatni i javni ključ kojim ćemo potpisivati i provjeravati validnost potpisanog dokumenta odnosno poruke. Možemo vidjeti serijaliziranu strukturu privatnog i javnog ključa. Primjećujemo da su znatno kraći od standarnih RSA ključeva.

```
-----BEGIN EC PRIVATE KEY-----
MIGkAgEBBDDivn3nlrpht6nVHp7Na1Ao9w31NzRLRJ+1FIt+hzIC1vblBGUCLmQ6+wi+7YJI2fSg
BwYFK4EEACKhZANiAARNun8OgzCjiZ6NEOLJsMkjGICQ9BFVcgxGtYSmqhRmcxICvj0n92BgnjmC
1dOs5AcTJQOnI8jL6/pIXhiyn06qdGieTmS1UubNLiKMbyccdufdmPEP33j8tUKLoPo4NsM=
-----END EC PRIVATE KEY-----
```

```
-----BEGIN PUBLIC KEY-----
MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAETbp/NIMwo4mejRDiybDJIxiAkPQRVXIMRrWEpqoUZnMS
Ar49J/dgYJ45gtXTrOQHEyUDpyPIy+v6SF4Ysp9OqnRonk5ktVLmzS4ijG8nHHbn3ZjxD994/LVC
i6D6ODbD
-----END PUBLIC KEY-----
```



Slika 2: Rezultat provjere

Korisnik će pokretanjem skripte generirati svoje ključeve te će izraditi zahtjev za potpisivanje certifikata.

Nakon generiranja privatnog i javnog ključa, potrebno je generirati certifikat kako bi digitalni potpis bio pouzdan i prepoznatljiv drugim stranama. Certifikat je digitalna potvrda koja povezuje identitet korisnika (npr. osobe ili organizacije) s njegovim javnim ključem.

Korisnik kreira zahtjev za potpisivanje certifikata CSR (Certificate Signing Request) u kojem se nalaze podaci koji će biti u certifikatu. Unutar openssl_csr.cnf definirali smo podatke o korisniku.

```
openssl req -new -key private_key.pem -out user_csr.pem -config openssl.cnf
```

Konačno rješenje trebalo bi za kranjeg korisnika pojednostavniti generiranje ključeva i CSR kroz grafičko korisničko sučelje (GUI).

Sada CA odnosno ustanova SUM, na osnovu zahtjeva za potpis (CSR) izdaje certifikat za određenog korisnika koji u ovom slučaju traje godinu dana. Kako bi korisnik imao mogućnost digitalnog potpisa, potrebno je kreirati extension file s potrebnim parametrima (digitalSignature, nonRepudiation, emailProtection, clientAuth).

```
openssl x509 -req -in user_csr.pem -CA ca_cert.pem -CAkey ca_private_key.pem
-CAcreateserial -out user_cert.pem -days 365 -sha256 -extfile openssl_csr.cnf
-extensions usr_cert
```

Korisnik sada posjeduje vlastiti certifikat čija se validnost može provjeriti naredbom.

```
openssl verify -CAfile ca_cert.pem user_cert.pem
```

```
robert@fedora:~/digitalni_potpis_sum$ python provjera.py
Signer info
-----------
Certificate subject: "Common Name: Robert Rozic, Organization: Sveuciliste u Mostaru, Locality: Mostar, Country: BA"
Certificate SHA1 fingerprint: 5259a49fdb86fd8ba483a775d985c27a2b888753
Certificate SHA256 fingerprint: edcff0a348878b305b611705978291e4031d5a340bf327790f91fc0d5d46452b
Trust anchor: "Common Name: sum.ba, Organization: Sveuciliste u Mostaru, Locality: Mostar, State/Province: Hercegovina, Country: BA"
The signer's certificate is untrusted.


Integrity
---------
The signature is cryptographically unsound.

The digest algorithm used was 'sha384'.
The signature mechanism used was 'sha384_ecdsa'.
The elliptic curve used for the signer's ECDSA public key was 'secp384r1' (OID: 1.3.132.0.34).


Signing time
------------
Signing time as reported by signer: 2024-09-10T11:53:20+00:00


Modifications
-------------
The signature does not cover the entire file.
Some modifications may be illegitimate, and they appear to be incompatible with the current document modification policy.


Bottom line
-----------
The signature is judged INVALID.
```

Slika 3: Rezultat provjere nakon izmjene

Na kraju kreiramo .p12 datoteku odnosno PKCS#12 datoteku. Ta datoteka je format za spremanje kriptografskih podataka, poput privatnih ključeva, javnih ključeva i X.509 certifikata. Ova datoteka je često zaštićena lozinkom kako bi se osigurala privatnost privatnog ključa.

U tu datoteku spremamo naš privatni ključ i certifikat koji ce nam kasnije služiti za potpisivanje datoteka. Kreiramo je naredbom:

```
openssl pkcs12 -export -out certificate.p12 -inkey private_key.pem
-in user_cert.pem -name "SUM certifikat" -passout pass:sum123
```

Nakon ovog koraka korisnik uspješno posjeduje certifikat za digitalno potpisivanje.

3.4 Digitalno potpisivanje dokumenta

Sljedeći korak je digitalno potpisivanje određenog dokumenta. Primjer je potpis testne diplome u PDF obliku. Kako bi potpisali PDF dokument koristeći Python skriptu, koristit ćemo pyhanko biblioteku.[11]

```
pip install pyhanko
```

```
1 from pyhanko.sign.fields import SigFieldSpec, append_signature_field
2 from pyhanko.pdf_utils.incremental_writer import IncrementalPdfFileWriter
3 from pyhanko.sign.signers import SimpleSigner, PdfSigner, PdfSignatureMetadata
4
5 signer = SimpleSigner.load_pkcs12(
6     pfx_file='certificate.p12',
7     passphrase=b'sum123'
```

```
)

with open("diploma.pdf", "rb") as doc:
    writer = IncrementalPdfFileWriter(doc)
    append_signature_field(writer, SigFieldSpec(sig_field_name="Signature1"))
    with open("signed_diploma.pdf", "wb") as signed_pdf_file:
        PdfSigner(
            signature_meta=PdfSignatureMetadata(field_name="Signature1"),
            signer=signer
        ).sign_pdf(writer, output=signed_pdf_file)
```

Konačno rješenje bi trebalo implementirati vizualni potpis koji bi se umetnuo na jednu ili više stranica PDF datoteke u obliku npr. QR koda koji vodi na link za provjeru potpisa.

3.5 Provjera valjanosti potpisa

Posljednji korak implementacije je provjera valjanosti potpisa.

```
from pyhanko.keys import load_cert_from_pemder
from pyhanko_certvalidator import ValidationContext
from pyhanko.pdf_utils.reader import PdfFileReader
from pyhanko.sign.validation import validate_pdf_signature

root_cert = load_cert_from_pemder('ca_cert.pem')
vc = ValidationContext(trust_roots=[root_cert])

with open('signed_diploma.pdf', 'rb') as doc:
    r = PdfFileReader(doc)
    sig = r.embedded_signatures[0]
    status = validate_pdf_signature(sig, vc)
    print(status.pretty_print_details())
```

Detalji o provjeri sadrže informacije o potpisniku, integritetu potpisa, vrijeme potpisivanja, izmjene na datoteci i zaključak. (Slika 2)

U svrhu testiranja provjere validnosti, ručno smo pomoću online alata promijenili jedan metapodatak unutar potpisane pdf datoteke kako bi dobili drugačije rezultate.(Slika 3)

4 Zaključak i daljnja implementacija

Primjena eliptične kriptografije (ECC) u procesu digitalnog potpisivanja akademskih dokumenata na Sveučilištu u Mostaru predstavlja značajan iskorak u povećanju sigurnosti i integriteta digitalnih zapisa.

Zbog svoje učinkovitosti s kraćim ključevima, ECC nudi bolju zaštitu od kriptografskih napada, uz manju potrošnju resursa. Suradnja sa Certiliom i AKD-om od velike je važnosti jer nam mogu prenijeti znanja koja su već utemeljena na funkcionalnim rješenjima ovog tipa.

Nastavak implementacije usmjerit će se na razvoj korisničkog sučelja za administratore, koji će moći kreirati i upravljati certifikatima. Dodatno bi bio razvijen API za potpisivanje dokumenata kako bi omogućili sigurno potpisivanje i autentifikaciju korisnika s više različitih aplikacija.

Aplikacija će imati integrirano sučelje za unos i pregled dokumenata koji se potpisuju, kao i opciju za dodavanje vizualnog potpisa i QR koda na potpisane dokumente.

Sustav za provjeru potpisa bit će dostupan putem web stranice, gdje će korisnici moći provjeriti autentičnost potpisa skeniranjem QR koda.

Literatura

[1] "SUMIT - Početna — sumit.sum.ba." https://sumit.sum.ba/. [Accessed 10-09-2024].

[2] "Common Cryptographic Architecture (CCA): PKA key algorithms — ibm.com." https://www.ibm.com/docs/en/linux-on-systems?topic=verbs-pka-key-algorithms. [Accessed 10-09-2024].

[3] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[4] V. S. Miller, "Use of elliptic curves in cryptography," in *Conference on the theory and application of cryptographic techniques*, pp. 417–426, Springer, 1985.

[5] "The Case for Elliptic Curve Cryptography - NSA/CSS — web.archive.org." https://web.archive.org/web/20090117023500/http://www.nsa.gov/business/programs/elliptic_curve.shtml. [Accessed 06-09-2024].

[6] "NSA Suite B Cryptography - NSA/CSS — web.archive.org." https://web.archive.org/web/20090207005135/http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml. [Accessed 10-09-2024].

[7] *Digital signature standard (DSS)*. 2013.

[8] *Digital Signature Standard (DSS)*. Feb. 2023.

[9] "Blockchain - Elliptic Curve Cryptography - GeeksforGeeks — geeksforgeeks.org." https://www.geeksforgeeks.org/blockchain-elliptic-curve-cryptography/. [Accessed 10-09-2024].

[10] "ecdsa — pypi.org." https://pypi.org/project/ecdsa/. [Accessed 06-09-2024].

[11] "pyHanko &x2014; pyHanko 0.25.2.dev1 documentation — pyhanko.readthedocs.io." https://pyhanko.readthedocs.io/en/latest/. [Accessed 09-09-2024].