Ministerul Educației, Culturii și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Raport

pentru lucrare de laborator Nr. 6
la cursul Criptografia și Securitate
"*Funcții Hash și Semnături Digitale*"

A efectuat: Alexei Ciumac, FAF-212
A verificat: Cătălin Mîțu

Chișinău - 2023

Subject: Study of hash based digital signatures for asymmetric ciphers

Tasks

Sarcina 1. *Studiați materiale didactice recomandate la temă plasate pe ELSE.*

Sarcina 2. *Utilizând platforma wolframalpha.com sau aplicația WolframMathematica, generați cheile, realizați semnarea și validarea semnaturii digitale a mesajului m pe care l-ați obținut realizând lucrarea de laborator nr. 2.Semnarea va fi realizată aplicând semnătura RSA. Valoarea lui n trebuie să fie de cel puțin 3072 biți. Algoritmul hash va fi selectat din lista de mai jos, în conformitate cu formula i = (k mod 24) +1, unde k este numărul de ordine al studentului în lista grupei, i este indicele funcției hash din listă: … (7 mod 24) + 1 = 8 ~ RipeMD-160.*

Sarcina 3. *Utilizând platforma wolframalpha.com sau aplicația Wolfram Mathematica, realizați semnarea și validarea semnăturii digitale a mesajului m pe care l-ați obținut realizând lucrarea de laborator nr. 2. Semnarea va fi realizată aplicând semnătura ElGamal (p și generatorul sunt dați mai jos). Algoritm hash va fi selectat …*

Theoretical notes

Within the realm of asymmetric cryptography, hash functions and digital signatures assume pivotal roles in establishing secure communication, ensuring data integrity, and facilitating authentication. Hash functions function as crucial tools, generating fixed-size representations—referred to as hash values or message digests—for variable-length data. In the domain of RSA (Rivest-Shamir-Adleman), a widely employed asymmetric encryption algorithm, hash functions play an integral role in the implementation of digital signatures.

In the RSA context, the sender employs a hash function on the message, resulting in a hash value subsequently encrypted with the sender's private key. Verification of the signature by the recipient, using the sender's public key, serves to confirm both the origin and integrity of the transmitted message. This process establishes a foundation of trust and safeguards against unauthorized modifications during transmission.

Likewise, the ElGamal asymmetric encryption algorithm incorporates hash functions into its digital signature scheme. In ElGamal's signature scheme, a hash function compresses the message, and the resulting hash value undergoes combination with mathematical operations involving the signer's private key and random numbers. This orchestrated process yields a distinctive digital signature that can be authenticated using the signer's public key.

The synergy of hash functions and digital signatures in asymmetric cryptography creates a resilient framework for secure communication. This framework empowers users to exchange information with confidence, assured of the authenticity and integrity of the transmitted data.

To perform this lab, I did not use Wolfram Alpha, but the functionality of the Python libraries: cryptodome and sympy.

RSA. The main idea of the RSA signature scheme is to use the same key generation as RSA encryption. To generate the signature we hash the original message and "encrypt" it not with the receiver's public key but rather with our private key so later we could be identified by "decrypting" the message with our (sender's) public key that will result in the same string as the decrypted message hashed.

```
def sign(self, msg, transmission):
      decimal_str =  int(''.join(str(ord(char)) for char in msg))

      hasher = RIPEMD160.new()
      hasher.update(str.encode(str(decimal_str)))
      hash_hex = hasher.hexdigest()
      hash_dec = int(''.join(str(ord(char)) for char in hash_hex))

      return (
            transmission,
            pow(hash_dec, self.private_key, self.public_key[0])
      )

def verify(self, transmission, sender_public_key):
      x = self.decrypt(transmission[0])
      s = pow(
            transmission[1],
            sender_public_key[1],
            sender_public_key[0]
      )

      hasher = RIPEMD160.new()
      hasher.update(str.encode(str(x)))
      hash_hex = hasher.hexdigest()
      hash_dec = int(''.join(str(ord(char)) for char in hash_hex))

      return hash_dec == s
```

Message requested in the task formulation was "Alexei CIUMAC", which when turned into an integer will be:

651081011201011053267738577 6567

After hashing the original message with RipeMD-160:

03f714bad74154733dfe89f097596676f48850da,

and bringing it to some integer format we get:
48511025549529897100555249535255515110010210156571024857555357545455541025256565348100097

And the signature produced by hash_dec ^ d mod n is:

331021364372578752906175013316503504248818577495458563315673321384725519386373503411
292010008951042770972826682802401260692489989144381594808380295203676059252704589819
152653945129099201432164663963062618865672230758157779357634173623217663383159566689
235317442542827807124647297141637085104783656835835350057581501320036739985199702495
065004056438278230332714785728750275795182958140181482323097130690654295681613891436
900406634570485067463525932382298469693818109447702239003446499653710750739159468250
441656361548986282183501299275098989128935678833100261397298087290947969450852113577
151036465731366314974932257081302655309623098040825359385536315114662436086491057355
250976152809062338870826519758018594014367841654227752116009400533442870809603128383
649231746992673031471338177061676367373701347578784301821525228776028224548397965169
571451192624518306667780659018625188429800995640794490024520569697195562749714762765

ElGamal. With this cryptosystem it is a bit trickier. The initial setup is the same. After the masking key and cryptogram y that make up the encrypted transmission are obtained, we compute the signature as follows:

s = k^(-1) (hash(msg) - dr) (mod p - 1), where:

- k is a secret random int such that GCD(k, p - 1) = 1 and
- r = g ^ k mod p

The signed message will be the triplet - (m, r, s). In order to verify this signature only public information (sender's: p, g, e) is need to compute:

v1 ≡ e ^ (r) * r ^ (s) mod p and v2 ≡ g ^ (hash(msg)) mod p,

if v1 ≡ v2 (mod p) the signature is declared valid.

```
def sign(self, msg, transmission):
    decimal_str =  int(''.join(str(ord(char)) for char in msg))



    k = self.generate_k()
    r = pow(self.public_key[1], k, self.public_key[0])



    hasher = RIPEMD160.new()
    hasher.update(str.encode(str(decimal_str)))
    hash_hex = hasher.hexdigest()
    hash_dec = int(''.join(str(ord(char)) for char in hash_hex))



    s = (mod_inverse(k, self.public_key[0] - 1) * (hash_dec - 
self.private_key * r)) % (self.public_key[0] - 1)
```

```
        return (transmission, r, s)

    def verify(self, transmission, sender_public_key):
        x = self.decrypt(transmission)


        hasher = RIPEMD160.new()
        hasher.update(str.encode(str(x)))
        hash_hex = hasher.hexdigest()
        hash_dec = int(''.join(str(ord(char)) for char in hash_hex))


        p1 = pow(sender_public_key[2], transmission[1],
sender_public_key[0])
        p2 = pow(transmission[1], transmission[2], sender_public_key[0])
        v1 = (p1 * p2) % sender_public_key[0]


        v2 = pow(sender_public_key[1], hash_dec, sender_public_key[0])


        print("v1 = v2 =\n", v2)


        return v1 % sender_public_key[0] == v2 % sender_public_key[0]
```

The signature that I got after performing the described above calculations is a pair of numbers:

(2511871350672536452626053719900961422150891458480137113166390985524917459156020947223416265921789541305132917285501111929307534121111025544503345823105075667230612292065045695960668230849494051053103910133216028393114611107000909146633209241252088252024976397609801113949908249994252674324930423807578252756461563915431825924335656847069472320665794026751442324547542876081141739670153719061411413749772891933807003905134071317638437323515539851804823882510320274471580844785582754462032014375264328562283649888591966061709863395897070951886691708785986603620539983304061475661973845197077224130819523778419931839384610809908079875848695555046123738150068047959346367534293192621128569116677096945791108568209831435861463527967870738911742295739378749410576082405046375348581187847318831336590069709286932875166884508773793817132821946853430964153230859943981918102108171401896340472709904566910826706715077379264134112946073,

1889033420506541545167807131634546723307192130272643477154202197744136996452865986181119905974667999442443829286185585537337437229467079247579784996797680881110549355980217030728200314424372888954057216152424626368648150907265879851869137390351275803066812244376001899463754941575354565532424417327254818294178159943084620513003311206669607181078904867349000559042066965248623726327455317628018676917779487578355434123508216141370959583600257198854377706554798566257080721362615516935618125964821658941078734361350906821939766845096440432023877801897800863789036272624037322606405379259440764149596936401747658541987799711327678789558968058300679334733611201441436156738133947292729331418268961840722228925678287513180613387845355939744064204306708030964892075280548154702887101392928019326284498093723516165888800849630073985177775772835692830606398995111103102181186610943031335339965832832505846483772110839557690144759030)

Conclusion

In summary, this laboratory investigation delved into the core principles of hash-based digital signatures within the framework of asymmetric ciphers, with a specific focus on the RSA and ElGamal algorithms. Through hands-on exploration, I acquired valuable insights into the pivotal role played by hash functions in upholding data integrity, authenticity, and non-repudiation in secure communication. The RSA algorithm exemplified the robust utilization of hash functions in crafting digital signatures, where the message's hash is encrypted with the sender's private key and verified by recipients using the corresponding public key. Likewise, the ElGamal algorithm demonstrated the seamless integration of hash functions in its signature scheme, enhancing the security and efficiency of the asymmetric cryptographic system.

Appreciating the implications of these principles is vital in understanding the resilience and dependability of hash-based digital signatures. These cryptographic techniques serve as indispensable tools in contemporary communication systems, offering a secure means to validate the origin and integrity of digital messages. As I delved into the application of hash functions in the RSA and ElGamal algorithms, I deepened my understanding of their role in preserving the confidentiality and trustworthiness of digital communication. Simultaneously, I acknowledged the significance of leveraging established libraries and adhering to best practices to ensure the utmost security in real-world applications.