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: Arteom KALAMAGHIN, FAF-211
A verificat: Aureliu ZGUREANU

Chișinău - 2023

Subject: Study of hash based digital signatures for asymmetric ciphers

Tasks

<u>Sarcina 1.</u> *Studiați materiale didactice recomandate la temă plasate pe ELSE.*

<u>Sarcina 2.</u> *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: ... (16 mod 24) + 1 = 17 ~ RipeMD-160.*

<u>Sarcina 3.</u> *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

In asymmetric cryptography, hash functions and digital signatures play pivotal roles in providing secure communication, data integrity, and authentication. Hash functions serve as essential tools for generating fixed-size representations of variable-length data, known as hash values or message digests. In the context of RSA (Rivest-Shamir-Adleman), a widely used asymmetric encryption algorithm, hash functions are integral to digital signatures. The sender applies a hash function to the message, producing a hash value that is then encrypted with the sender's private key. The recipient can verify the signature using the sender's public key, ensuring both the origin and integrity of the message. This process establishes trust and prevents unauthorized modifications during transmission.

Similarly, the ElGamal asymmetric encryption algorithm employs hash functions in its digital signature scheme. In ElGamal's signature scheme, a hash function is used to compress the message, and the hash value is then combined with mathematical operations on the signer's private key and random numbers. This process results in a unique digital signature that can be verified using the signer's public key. The combination of hash functions and digital signatures in asymmetric cryptography ensures a robust framework for secure communication, enabling users to exchange information with confidence in the authenticity and integrity of the transmitted data.

Implementation

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 “Arteom KALAMAGHIN”, which when turned into an integer will be:

651141161011111093275657665776571727378

After hashing the original message with RipeMD-160:

1df30f8c76ffe270766a51f4dca5c9c1c277b258,

and bringing it to some integer format we get:

491001025148102569955541021021015055485554549753491025210099975399579949995055559 8505356

And the signature produced by hash_dec ^ d mod n is:
25930275651337785472149619304485197064979482190825211012907179740586210305173796748332200202332123659379771395561312834853202056767933482602351988889115301554614058484939153347422282737635128158263179324157656670123878108389774890195289350165349400001260784519078566683493540607885267002584218677131424195921504114319705435755027193295552929213197919288175534959723297920235042802589734488074315612094334338133003656967718113661649745622764595745058825335199603698421047204342588152956617184825741923559948920536523910392720737204987096758785880538747599124894559018170451071281020935435540293249601232901327862642029012150049303701170329875006604658252669474522690772140362846930286941152803141697481358177241610935730773667547101784839040053384427312098032552519489926983747614096584680258972365194866398122362005284444539733001822778071476157283612441974522800973575860071751906061540287273474403161310016451669804561505 89

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))
    hash_dec = Null # Look for how it is made in the RSA

    k = self.generate_k()
    r = pow(self.public_key[1], k, self.public_key[0])
    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)
    hash_dec = Null # Look for how it is made in the RSA

    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])

    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:
(126803421859574431159254980217742171220211192302472742283377865741195356296807315
1547155261279580128163632771219216499676255095674826730477719938135988690482407594
8668216650559181678490249461335080540417717741175278317749375465411974264145526225
9303286954291698588675921516560673518332913814177414859095843247604362781113720762
5576470181295473232685584181190769395346528870083157881607619326032937236227392096
2748918485508655164500982411252440114681755792042880312177886624276089730949835229
3841319110800775847432661284756344344395061318893890108285969225823596911089055034
3107269458606354695726851799674148498555298593050195336502881913606678346548221114
9219871634897812743905806133156759888078240597920780072033618509575756975559731515
1754816089184719723503470482302572076424329679020607204087459238994951957419207968
5782280268550279612682436464886512225454191743276172392336180302799806949348773462
580876259422455591986801,
2802074846949504372611675136770143165193984207108475692617318917865825212989226150
0671296930480951347418155218381371914230741580850233209939032767620720222426664570
3259389722351680686747715517737718762144012147554196838320160402415896263484913370
8789788263519496557618025239261561216927604659341898992910821061613583334534807937
7449832369525453536618189609601522619395634208850504140150976680011043683168196819
5941078712528668999098476570781360554673188499400184995659197961492471620602519706
6713169325750656767489396422292901069828316456336887403424179205260545695439970989
9434939084170216300373922114036951479250099580153437069415271498287662704007106420
9229936509544827663000478909470330975010260851181746130015077670744838976811488080
8964178748636092349031249546485770311584100637994278096858999345699490958008151621
1779748349164305176042993861677499466074984866761120020296488773222846023280091596
8175946670432202300651)

Conclusion

In conclusion, this laboratory work delved into the fundamental principles of hash-based digital signatures within the context of asymmetric ciphers, specifically focusing on the RSA and ElGamal algorithms. Through practical exploration, I gained valuable insights into the critical role that hash functions play in ensuring data integrity, authenticity, and non-repudiation in secure communication. The RSA algorithm demonstrated the robust application of hash functions in creating digital signatures, where the hash of the message is encrypted with the sender's private key and verified by recipients using the corresponding public key. Similarly, the ElGamal algorithm showcased the integration of hash functions in its signature scheme, contributing to the security and efficiency of the asymmetric cryptographic system.

Understanding the implications of these principles is crucial in appreciating the strength and reliability of hash-based digital signatures. These cryptographic techniques serve as essential tools in modern communication systems, providing a secure means for verifying the origin and integrity of digital messages. As I explored the utilization of hash functions in the RSA and ElGamal algorithms, I gained a deeper appreciation for their role in upholding the confidentiality and trustworthiness of digital communication, while simultaneously recognizing the importance of using established libraries and best practices to ensure the highest level of security in real-world applications.

Check out my GitHub repo for the source code of these project:
https://github.com/Starlight-Crusader/CS-Lab