Исследование алгоритмов генерации и верификации ЭЦП

Выполнила: Макаров Алексей Игоревич, 3 курс 4 группа 2 подгруппа

2024

**Теоритические сведения**

Электронная цифровая подпись (ЭЦП) является важным элементом современных информационных систем, использующих методы и технологии криптографического преобразования информации.

Определение 1. Электронная цифровая подпись – контрольная характеристика сообщения, которая вырабатывается с использованием личного ключа, проверяется с использованием открытого ключа, служит для контроля целостности и подлинности сообщения и обеспечивает невозможность отказа от авторства.

Таким образом, ЭЦП выполняет те же функции, что и собственноручная (поставленная «от руки») подпись:

• аутентифицирование лица, подписавшего сообщение;

• контроль целостности подписанного сообщения;

• защита сообщения от подделок;

• доказательство авторства лица, подписавшего сообщение, если это лицо отрицает свое авторство.

Важнейшие отличительные особенности ЭЦП:

• ЭЦП представляет собой бинарную последовательность (в отличие от графического образа, каковым является подпись от руки);

• указанная бинарная последовательность зависит от содержания подписываемого сообщения.

Электронная цифровая подпись – бинарная (или в ином виде) последовательность символов, являющаяся реквизитом электронного документа, зависящая от содержания этого документа и предназначенная для подтверждения целостности и подлинности электронного документа.

Алгоритм DSA (Digital Signature Algorithm – алгоритм цифровой подписи), или DSS (Digital Signature Standard – стандарт цифровой подписи), является одним из известных, нередко и сейчас применяемых. В алгоритме используются следующие параметры: p – простое число длиной от 64 до 1024 битов (число должно быть кратно 64); q – 160-битный простой множитель (р – 1). Числа p, q, v могут использоваться группой лиц. Еще один элемент открытого ключа у вычисляется в соответствии с выражением:



ЭЦП Эль-Гамаля: Итак, ключевая информация отправителя: открытый ключ: y, g и р; тайный ключ: х. Чтобы подписать сообщение Мо, обладатель используемых для ЭЦП ключей должен выбрать, как и в предыдущей схеме, случайное число k, взаимно простое с (р – 1). Затем вычисляется числа а и b, являющиеся цифровой подписью (S = {a, b}):



Для верификации подписи вычисляется хеш полученного сообщения Н(Мп) = h. Далее нужно убедиться, что выполняется равенство:



ЭЦП Шнорра: Рассматриваемая схема является основой стандарта ЭЦП в Беларуси. Алгоритм ЭЦП К. Шнорра (К. Schnorr) является вариантом алгоритма ЭЦП Эль-Гамаля.

Выбирается число х < q (х является тайным ключом) и вычисляется последний элемент открытого ключа:



Секретный ключ имеет длину не менее 160 битов. Для подписи сообщения Мо выбирается случайное число k (1 < k < q) и вычисляет параметр а:



Далее вычисляется хеш от канкатенации сообщения Мо и числа а: h = H(Mo||a). Обратим внимание, что хэш-функция непосредственно не применяется к сообщению. Создается хеш-образ подписываемого сообщения, спереди присоединенного к числу а. Далее вычисляется значение b:



Для проверки подписи получатель вычисляет:



Основные вычисления для генерации подписи могут производиться предварительно. Порядок величин х и h – около 140 двоичных разрядов, порядок числа k – около 70–72 разрядов. С учетом этого сложность операций умножения можно считать ничтожно малой по сравнению с модульным умножением в схеме RSA.

**Ход работы**

Вариант 8

В данной работе было необходимо реализовывать авторское оконное приложение в соответствии с целью лабораторной работы.

Код программы представлен в листинге 1.

|  |
| --- |
| import time import hashlib from cryptography.hazmat.primitives.asymmetric import rsa, padding from cryptography.hazmat.primitives import hashes, serialization from concurrent.futures import ThreadPoolExecutor from Crypto.PublicKey import ElGamal from Crypto.Random import get\_random\_bytes from ecdsa import SigningKey, NIST192p  def rsa\_generate\_keys():  *private\_key* = rsa.generate\_private\_key(  public\_exponent=65537,  key\_size=2048  )  *public\_key* = *private\_key*.public\_key()  return *private\_key*, *public\_key* def rsa\_sign\_message(private\_key, message):  *signature* = private\_key.sign(  message,  padding.PSS(  mgf=padding.MGF1(hashes.SHA256()),  salt\_length=padding.PSS.MAX\_LENGTH  ),  hashes.SHA256()  )  return *signature* def rsa\_verify\_signature(public\_key, message, signature):  try:  public\_key.verify(  signature,  message,  padding.PSS(  mgf=padding.MGF1(hashes.SHA256()),  salt\_length=padding.PSS.MAX\_LENGTH  ),  hashes.SHA256()  )  return True  except:  return False    def elgamal\_generate\_keys():  *key* = ElGamal.generate(2048, get\_random\_bytes)  return *key*, *key*.publickey()  def elgamal\_sign\_message(private\_key, message):  *h* = int.from\_bytes(hashlib.sha256(message).digest(), byteorder='big')  *k* = private\_key.random\_element()  *signature* = private\_key.sign(*h*, *k*)  return *signature* def elgamal\_verify\_signature(public\_key, message, signature):  *h* = int.from\_bytes(hashlib.sha256(message).digest(), byteorder='big')  return public\_key.verify(*h*, signature)  def schnorr\_generate\_keys():  *sk* = SigningKey.generate(curve=NIST192p)  *vk* = *sk*.verifying\_key  return *sk*, *vk* def schnorr\_sign\_message(private\_key, message):  return private\_key.sign(message)  def schnorr\_verify\_signature(public\_key, message, signature):  return public\_key.verify(signature, message)  def measure\_time(func, \*args):  *start\_time* = time.time()  *result* = func(\*args)  *end\_time* = time.time()  return *result*, *end\_time* - *start\_time* def key\_exchange\_simulation():  *message* = b"Sample message for signing"   with ThreadPoolExecutor() as *executor*:  *rsa\_future* = *executor*.submit(rsa\_generate\_keys)  *elgamal\_future* = *executor*.submit(elgamal\_generate\_keys)  *schnorr\_future* = *executor*.submit(schnorr\_generate\_keys)   *rsa\_private*, *rsa\_public* = *rsa\_future*.result()  *elgamal\_private*, *elgamal\_public* = *elgamal\_future*.result()  *schnorr\_private*, *schnorr\_public* = *schnorr\_future*.result()   *rsa\_signature*, *rsa\_time* = measure\_time(rsa\_sign\_message, *rsa\_private*, *message*)  *rsa\_verification\_result*, *rsa\_verify\_time* = measure\_time(rsa\_verify\_signature, *rsa\_public*, *message*, *rsa\_signature*)   *elgamal\_signature*, *elgamal\_time* = measure\_time(elgamal\_sign\_message, *elgamal\_private*, *message*)  *elgamal\_verification\_result*, *elgamal\_verify\_time* = measure\_time(elgamal\_verify\_signature, *elgamal\_public*, *message*, *elgamal\_signature*)   *schnorr\_signature*, *schnorr\_time* = measure\_time(schnorr\_sign\_message, *schnorr\_private*, *message*)  *schnorr\_verification\_result*, *schnorr\_verify\_time* = measure\_time(schnorr\_verify\_signature, *schnorr\_public*, *message*, *schnorr\_signature*)   print(f"RSA: sign time {*rsa\_time*}, verify time {*rsa\_verify\_time*}")  print(f"ElGamal: sign time {*elgamal\_time*}, verify time {*elgamal\_verify\_time*}")  print(f"Schnorr: sign time {*schnorr\_time*}, verify time {*schnorr\_verify\_time*}")  key\_exchange\_simulation() |

Листинг 1 – Программная реализация приложения

Результат работы программы представлен в листинге 2:

|  |
| --- |
| RSA генерация: 0.0017669200897216797 seconds  RSA: b"F\xbdY\xa9\xe9\x8e\xaa\xe3\xd2\xc0^\x94\rZ\x1b\xdfy\xb9A\x00\xdd\xd5?4\x17C\x0bH\xf0\x7f\r\x9ef\x85L\xd4\xb8\xef\xf2j\xdb\xf3a(5'\xff\xc7\xf7\xc5\x14\x9fe\x0f\x8c\x0e\x1f!)\x81\xfd\xa2DE\xda\x06\x1bp\x8b\x92\xe6/\x04\xbc\xd5\xa3#\xdc8\xea\x8c\x03\xb3\x91.39&\x0fiu\x17\xcb.\xa7\x90\xd2/\nPNo\xf3\xd7\xe9\x85\xb9\x1a\x18sz\xbfs\xba\x96\x07\xf1\xcc\xba\x15\x97\xf1\xbep\xc1\xac\xce\xfa\x95W\x1a\xdf\x9a\x12\xe4&\xee\xdf\x804\xc3\xafm\xe5\xdc\x8f\xd7Ei\xa1\xb6B\xff\xf5\xf5{-\x10R\x134\x9a\x8d\x1eu\t\xdb\xcc\x7f4\xbbl\xff\x96\x11\xd1\xdfd\xd0L\xb4\x8ff!\x01!\x8c\x0fO\x15\xd8=\xe9\xd1\xdar\xee\xe4o\xcdR\x8fs\xaf\x8c\xc5\x9e]\xa8s\xd7\xc1N(\xd5\xe4\xb6\xfa\x90!\x7f\x0c8\xa1\x01uE\x11E\xe9}\xc4/\xf7^\xc5!A\xff\xe9\xe0CU\x1f\xf6\xf3\r\xae;\_\xdc\xa4\xd6\x8foL"  RSA верификация: 0.0002338886260986328 seconds  RSA: True  Эль-Гамаля генерация: 0.0004048347473144531 seconds  Эль-Гамаль : b'0e\x021\x00\x83Lw\x91Q\xaa\x03\xc7o\xdb\x97\xe5H\xb4\xb8\xb1\\\xd8\xe1\xbcj\xca\xff\xdb\xcb\x99\x996\x08\x89G6\xb6\xe0i\x8d\xd1\xe1\x7f?\xaf\x1eR\x97\x12\xe6\*\xec\x020R%\xe08O\x83\xcf\xcdDD^\x8e3]\x8d\xc6\xcc\xc5\x13\xec\x87\x11^\x10\x1b\x07@\xc4j\xf3\xec\xa2~\xe9\_9\xfdL\xc9\x0e\xab\xb6\xe2Y\xe1\xea\x15\x85'  Эль-Гамаля верификация: 0.0008058547973632812 seconds  Эль-Гамаль : True  Шнорра генерация: 0.0003845691680908203 seconds  Шнорр : b"0f\x021\x00\x90\xbd\xcc\xad4\xa0\x84R\xa6&\xff.\x0f\xedr\*^\xef\x97|,\x15uB\xe5\x90\_\x1c\xa1\x81\x15\xf0\xc4\xeeV\x8b\xff\xb0]\xd81\x0bT\xa8\xd5\xedg\xd0\x021\x00\xb6o\xa9\x8f\x95'T\xde\xc1C\*U\x8a\x96\xb0rN\xaf\xa9F\xa7^\xa3\xd3~\xeal9\xc4\x8db\xd8\xc6\x8fS\x08>\xa3\xe9\xbe\x06\_\xc4@\r9\xd7\xe5"  Шнорра верификация: 0.0007524490356445312 seconds  Шнорр : True |

Листинг 2 – Результат работы приложения

**Вывод**: таким образом, были закреплены теоретические знания по алгебраическому описанию и алгоритмам реализации операций генерации и верификации электронной цифровой подписи (ЭЦП).