MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA
TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Criptography and Security

Laboratory work 6: Hash functions and digital signatures

Elaborated:
st.gr. FAF-211 Cristian Brinza
Verified:
asist.univ. Cătălin Mîțu

Chișinău, 2023

RSA Algorithm

The RSA algorithm is one of the first public-key cryptosystems and is widely used for secure data transmission. Named after its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman, it was introduced in 1977. RSA is based on the mathematical difficulty of factoring large integers, which is the product of two large prime numbers. The RSA algorithm involves four steps: key generation, key distribution, encryption, and decryption.

Key Generation: RSA begins by selecting two large prime numbers, p and q, and calculating their product $N = pq$, which is called the modulus. The totient of N is computed as $\phi(N) = (p-1)(q-1)$. Then, an integer e is chosen such that $1 < e < \phi(N)$ and e is coprime to $\phi(N)$. The pair (e,N) serves as the public key. The private key is computed as d, which is the modular multiplicative inverse of e modulo $\phi(N)$. Key Distribution: The public key (e,N) is distributed openly, while the private key d is kept secret.

Encryption: To encrypt a message M, the sender computes the ciphertext C using the recipient's public key with $C = M^e \mod N$.

Decryption: The recipient can decrypt the ciphertext C using their private key d with $M = C^d \mod N$, recovering the original message M. RSA's security relies on the computational difficulty of the integer factorization problem, and as such, the keys used in RSA encryption need to be sufficiently large to prevent factorization by modern computing methods.

ElGamal Encryption

ElGamal encryption is a public-key cryptosystem developed by Taher Elgamal in 1985. It is based on the Diffie-Hellman key exchange and uses the discrete logarithm problem as its foundation, which is considered difficult to solve. ElGamal encryption consists of three components: key generation, encryption, and decryption.

Key Generation: The user generates a key pair consisting of a private key x and a public key (p,g,h), where p is a large prime number, g is a primitive root modulo p, and $h = g^x \mod p$.

Encryption: To encrypt a message M, the sender selects a random integer k and computes the shared secret $s = h^k \mod p$. The ciphertext is then a pair (C_1, C_2), where $C_1 = g^k \mod p$ and $C_2 = M \cdot s \mod p$.

Decryption: The recipient uses their private key x to compute $s = C_1^x \mod p$ and then retrieves the message M by calculating $M = C_2 \cdot s^{-1} \mod p$.

ElGamal is unique in that it provides an element of randomness in the encryption process, which results in different ciphertexts for the same plaintext message when encrypted multiple times.

Digital signatures

Digital signatures are a crucial component in the realm of digital security, functioning as the electronic equivalent of physical signatures or seals, but with enhanced security features. Rooted in the principles of public key cryptography, digital signatures employ a pair of keys: a public key, which is openly shared, and a private key, which remains confidential with the owner. The process begins with the creation of a hash, a unique fixed-size string, from the original document using a hash function. This hash undergoes encryption with the signer's private key, forming the digital signature. This signature is then attached to the document. Upon receipt, the recipient decrypts the signature using the signer's public key, retrieving the hash value initially encrypted by the sender. Simultaneously, the recipient generates a new hash from the received document. If this newly generated hash matches the one obtained from decrypting the signature, it confirms two essential aspects: the document's integrity, assuring it hasn't been altered post-signing, and the authenticity of the signature, verifying it was indeed signed by the holder of the private key. This mechanism not only guarantees the document's authenticity but also provides non-repudiation, ensuring the signer cannot deny their association with the document. Digital signatures are widely applied in various sectors, including secure email communications, software distribution, financial transactions, and legal documentation, providing a layer of security and trust in digital interactions.

SHA-384

SHA-384 is part of the SHA-2 (Secure Hash Algorithm 2) family. It generates a 384-bit (48-byte) hash value, often rendered as a hexadecimal number:

1. **Input Data**: Any size of data.
2. **Hashing**: The data is processed through the SHA-384 algorithm, producing a fixed-size 384-bit hash.
3. **Uniqueness**: Any change in the data results in a substantially different hash.

RSA with SHA-384 for Digital Signatures:

1. **Hash the Message**: First, the sender hashes their message using SHA-384. This produces a unique, fixed-size hash.
2. **Encrypt the Hash with RSA Private Key**: The sender then encrypts the hash with their RSA private key, creating the digital signature. This process is unique to the sender (due to their private key).
3. **Send Message and Signature**: Both the original message and its signature are sent to the recipient.

Verification Process

1. **Decrypt the Signature**: The recipient uses the sender's public RSA key to decrypt the digital signature. This reveals the hash value.
2. **Hash the Received Message**: The recipient hashes the message they received using the same SHA-384 algorithm.
3. **Compare Hashes**: If the hash from the decrypted signature matches the hash of the received message, the signature is verified. This ensures that the message was indeed sent by the owner of the private key and has not been altered.

Key Generation

1. **Choose a Large Prime Number** :Select a large prime number.
2. **Choose a Generator** :Select a generator g which is a primitive root modulo p.
3. **Choose a Private Key** : Randomly choose x such that 1 ¡ x ¡ p-1.
4. **Compute the Public Key** :Calculate y = gx̂ mod p.The public key is (p, g, y).

Signature Generation

1. **Hash the Message:**Hash the message M using SHA-256 to produce a hash value H(M).
2. **Choose a Random Integer:** Select a random integer k such that 1 ¡ k ¡ p-1 and $\gcd(k, p-1) = 1$.
3. **Compute r:** Calculate r = gk̂ mod p.
4. **Compute s:** Solve for s in the congruence H(M) = x $\cdot r + k \cdot s$ mod (p-1).

Signature Verification

1. **Compute** w : Calculate w = s^1 mod (p-1).
2. **Compute** u_1:Calculate u_1 = H(M) $\cdot w$ mod (p-1).
3. **Compute** u_2:Calculate u_2 = r $\cdot$ w mod (p-1).
4. **Verify the Signature**:The signature is valid if gĤ(M) $\equiv$ yr̂ $\cdot$ rŝ mod p.

Security Considerations

The security of the ElGamal signature scheme relies on the difficulty of solving the discrete logarithm problem in the group defined by pp and gg. The hash function SHA-256 ensures the integrity of the message by producing a unique hash for every unique input, making it computationally infeasible to find two different messages with the same hash.

Implementation

Task 2: RSA Encryption Algorithm Implementation with Digital Signature using SHA-384

The code starts by generating two large prime numbers, p and q, each with a bit length of 1536, using the generateprimenumber function. These primes are used to calculate the modulus n = p * q, which is a key component of the RSA algorithm. The totient of n, denoted as phi, is computed as (p - 1) * (q - 1), and a standard public exponent e (commonly 65537) is chosen. The private exponent d, which is crucial for creating the digital signature, is then calculated using the extended Euclidean algorithm (extendedgcd) to find the modular inverse of e modulo phi. For signing a message, the message is first hashed using SHA-384, a strong cryptographic hash function, to generate a fixed-size hash value. This hash is then encrypted using the private key d through modular exponentiation (modexp), producing a digital signature unique to the message and the private key. When verifying the signature, the same hashing process is applied to the original message. The signature is then decrypted using the public exponent e. If the decrypted signature matches the hash of the original message, it confirms the signature's validity, ensuring the message's integrity and authenticity. This process not only secures the message against tampering but also verifies the identity of the sender, as only the holder of the private key could have generated the valid signature.

Task 3: ElGamal Encryption Algorithm Implementation with Digital Signature using SHA-256

The code begins by defining a large prime number p and a base g (a generator of the multiplicative group of integers modulo p). The genkeys function generates a pair of keys: a private key (a random integer kpr) and a corresponding public key consisting of p, g, and e (where e is g raised to the power of the private key modulo p). For signing a message, the getsignature function first selects a random integer kE that is relatively prime to p-1. It then computes r, which is g raised to the power of kE modulo p. The SHA-256 hash of the message is calculated, and using kEinv (the modular multiplicative inverse of kE), the signature S is computed. This signature is a tuple (r, S), unique to the message and the private key. To verify the signature, the verifysignature function recalculates the hashed message using parts of the public key and the signature. It also computes the SHA-256 hash of the message independently. If these two hashed values match, it confirms the integrity and authenticity of the message, demonstrating that the message was signed with the corresponding private key and has not been altered. This process effectively uses the ElGamal signature scheme to ensure the security of digital communications, leveraging the difficulty of discrete logarithm problems and the reliability of SHA-256 hashing.

Conclusion

In conclusion, this laboratory work, centered on implementing RSA and ElGamal digital signature systems with SHA-384 and SHA-256 hashing functions, provided a practical and in-depth exploration of cryptographic principles and their application in digital security. Through hands-on experience, it illuminated the complex mechanisms underlying public-key cryptography, particularly highlighting the significance of prime number-based algorithms and modular arithmetic. The integration of robust hashing algorithms like SHA-384 and SHA-256 underscored the crucial role of data integrity and authenticity in securing digital communications. Key generation and management, a fundamental aspect of cryptography, was effectively demonstrated, emphasizing the importance of private key security and the public dissemination of public keys. This lab work not only reinforced the theoretical concepts of digital signatures, ensuring integrity, authenticity, and non-repudiation but also shed light on the broader security implications inherent in cryptographic systems. By bridging theoretical knowledge with practical application, the laboratory work offered essential insights into the real-world applications of cryptography, preparing participants for various challenges in the field of computer security and digital communication.

https://github.com/CristianBrinza/UTM/tree/main/year3/cs/lab6