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 5: Cryptography with Public Keys

Elaborated:
st.gr. FAF-211 Coreţchi Mihai
Verified:
asist.univ. Cătălin Mîţu

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.

AES Algorithm

The Advanced Encryption Standard (AES) is a symmetric key encryption algorithm established as an encryption standard by the U.S. National Institute of Standards and Technology (NIST) in 2001. AES is a variant of the Rijndael cipher developed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen.

AES operates on a fixed block size of 128 bits and uses keys of 128, 192, or 256 bits, known as AES-128, AES-192, and AES-256, respectively. The AES encryption process involves several rounds of processing for each block of data, with the number of rounds depending on the key size: 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys.

Each round consists of four stages:

1. **SubBytes**: A non-linear substitution step where each byte is replaced with another according to a lookup table.
2. **ShiftRows**: A transposition step where each row of the state is shifted cyclically a certain number of steps.
3. **MixColumns**: A mixing operation which operates on the columns of the state, combining the four bytes in each column.
4. **AddRoundKey**: A simple bitwise XOR of the current block with a portion of the expanded key.

Before these rounds, an initial AddRoundKey stage is applied. After the final round, a slightly modified set of operations is applied to produce the ciphertext.

AES is widely adopted across the globe and is used in various applications to protect sensitive data due to its strength and efficiency in a wide range of hardware and software environments.

Implementation

Task 2.1: RSA Encryption Algorithm Implementation

In Task 2.1, the RSA encryption algorithm, essential for secure data transmission, is implemented as follows:

1. **Prime Number Generation**: Two large prime numbers p and q are generated using the `generate_prime_numbe` function, which ensures the numbers have a specified number of bits for security.
2. **Calculation of n and $\phi(n)$**: The values of $n = p \times q$ and Euler's totient function $\phi(n) = (p-1) \times (q-1)$ are computed.
3. **Choosing Encryption Exponent**: An encryption exponent e, typically 65537 for its efficient computational properties, is chosen.
4. **Computing Decryption Exponent**: The decryption exponent d is the modular inverse of e modulo $\phi(n)$, computed using the Extended Euclidean Algorithm implemented in `extended_gcd`.
5. **Encryption**: The message is raised to the power of e modulo n, using the function `mod_exp` for efficient modular exponentiation.
6. **Decryption**: The inverse operation is performed, where the encrypted message is raised to the power of d modulo n, converting it back to the original message.

This process ensures secure communication, as only the holder of the private key d can decrypt a message encrypted with the public key e.

Result:

Encrypted Message: 101262077782564725644673097882028498833964245923465512637451794961578263931244373232797145533133223275770476983507031328352353345271767339947678624372427068394995836371302880789510289050253395801584433565332849671183906800950258741524612492993448865830238010094786392287613430126104105976817852103253494829664523638739279649460965720085469019429164543151173532870579459019241649713268490411573493101004445021130546738000264255677585579108040102884586069363800518083466337219429565949439207688642215211234393193691895327639313465980593810264794057725776371954181124383102142230903131478615115381570329753732570002435163460741777291840120859350658523784480217785140845932597929072655630518497863479034428411719901334040212611854228747819087674931903609482418301327433634009790677064341032034671256790072205550408776868578577609744096732730810476787690077941818404932845213610216724247040619676982248795971327145604153158892348869034169338688966544831671723944952469290962533709337120280222550035145631682800448756762452750053221396973001992584936254670034291184126 1

92645261793945507428588986420937023694859842532543664277402906326817403835029897487805673
28176223112959350171776500887248942294819017593592499552837340927286870659052955
Decrypted Text: Coretchi Mihai

Task 2.2: ElGamal Encryption Algorithm Implementation

Task 2.2 demonstrates the implementation of the ElGamal encryption algorithm, as follows:

1. **Modular Exponentiation**: A function `mod_exp` is defined to perform modular exponentiation efficiently, which is essential for both encryption and decryption processes in ElGamal.
2. **ElGamal Encryption Function**: The `encrypt` function takes a message, public parameters p and g, and the recipient's public key y. It generates a random integer k, calculates $c1 = g^k \mod p$ and $c2 = (\text{message} \times y^k) \mod p$, and returns $c1$ and $c2$ as the encrypted message.
3. **ElGamal Decryption Function**: The `decrypt` function takes the encrypted message components $c1$ and $c2$, the modulus p, and the recipient's private key x. It computes the original message as $(c2 \times c1^{p-1-x}) \mod p$.
4. **Key Generation**: Public parameters p (a large prime number) and g (a generator) are defined. The private key x is a randomly chosen integer, and the public key y is computed as $y = g^x \mod p$.
5. **Message Encryption**: The message "Coretchi Mihai" is converted to its decimal representation. Each character of the message is encrypted using the `encrypt` function, resulting in an array of encrypted tuples.
6. **Message Decryption**: Each tuple in the encrypted message array is decrypted using the `decrypt` function, reconstructing the original message in its decimal form.

This implementation of the ElGamal encryption algorithm illustrates the process of secure message encryption and decryption, leveraging the difficulty of discrete logarithm problems in cryptography.

Result:

Coretchi Mihai in Decimal Form: [67, 111, 114, 101, 116, 99, 104, 105, 32, 77, 105, 104, 97, 105]

Decrypted Message in Decimal Form: [67, 111, 114, 101, 116, 99, 104, 105, 32, 77, 105, 104, 97, 105]

Encrypted Message:

[(10587513916305534901130316508141853680927700346123770541798909832070241437794785889789241968564382037731237634662670892268592651031729565458662792071045409728740287406211694712727994056539585771010440722659002182527210866494241474471279004692889419032507646084101750873033099900824742089881879102650130453590712405657893547224360861755270494337707937258239918610704567460981682100951611554125701517248288071565365597157687463224607554133076193709350199250901902333164458579970758296339036203624408915847200840102255707043368791978479458667816878997723151873557624228727135966237673009956746562116606668096067282563757,

248263722076403947292013074976897928442526424809094713089421289108706103280962033893734292394103046336030476411326740062036042757662558080544084892185024573083721049079670554203689039287609150337435445053969403904765572453682087058878019767227389498861289696713302594194114200725485796378373147560606103709530129031620920034331870074984526982779466000006495809976289337909491946283298444760839865912790187432821014382553569620581426499748667202664768694164991098010408399429183778472790445544119103460982214494063181078402088476618310228620992396421448864879377345864034236950284907705059960034617651602397841661598921), ...

Task 3: Diffie-Hellman Key Exchange Algorithm Implementation

In Task 3, the implementation of the Diffie-Hellman key exchange algorithm is demonstrated. This algorithm is essential for securely exchanging cryptographic keys over a public channel. The process involves the following steps:

1. **Modular Exponentiation Function**: The `mod_exp` function performs modular exponentiation, a crucial operation in the Diffie-Hellman algorithm for calculating public and common keys.
2. **Parameter Definition**: Public parameters p (a large prime number) and g (a primitive root modulo p) are defined.
3. **Secret Numbers Generation**: Secret integers a and b are randomly generated for Alice and Bob, respectively.
4. **Public Key Computation**: Alice's public key A is computed as $A = g^a \mod p$, and Bob's public key B is computed as $B = g^b \mod p$.
5. **Common Key Calculation**: Alice computes a common key using Bob's public key B as common_key_alice $= B^a \mod p$. Similarly, Bob computes the common key using Alice's public key A as common_key_bob $= A^b \mod p$.
6. **Key Verification**: It is verified that both computed keys by Alice and Bob are identical, ensuring a successful key exchange.

This implementation demonstrates the fundamental principles of the Diffie-Hellman key exchange, highlighting its utility in establishing a shared secret over an insecure communication channel.

Result:

Alice's Common Key: 20534920197706640796755848011697562486639880747672533952640638731 2

Bob's Common Key: 20534920197706640796755848011697562486639880747672533952640638731 2

Keys are identical: True

Conclusion

Finally, the RSA, ElGamal, and AES algorithms remain crucial cryptography algorithms in the digital era. Factorization of large primes is the basis for strong public-key encryption and digital signatures of RSA. ElGamal's use of the discrete logarithm problem presents a safe alternative to asymmetric encryption with the added bonus of cryptographic randomness. However, AES serves as the gold standard for symmetric key encryption, offering a reliable means of safeguarding data confidentiality globally.

The algorithms show just how carefully the mathematical complexities combine with computational feasibility, giving a clue to why modern cryptography is critical to digital security. Due to the increased complexity of threats to information security, the importance of these cryptographic algorithms cannot be understated. First, it is important to realize that such algorithms are not only important for securing communication and data but also remain a driving force for the development of new cryptographic techniques and the improvement of existing ones.

Cryptography continues to undergo evolution to ensure that it can address the security needs of the next generation of technologies. This will therefore be an important area of research in the fight against the ever-changing cyber threats, such as cryptographic algorithms like RSA, ElGamal and AES.

https://github.com/eamtcPROG/CS/tree/main/Lab5