Portfolio Part 2

CSI2108

Jordan Farrow (10653054)

Edith Cowan University, Joondalup

Dr. Julia Collins

Table of Contents

1. RSA Key Pair Generation

For secure key exchange to occur, RSA key pairs will be generated and implemented.

1.1 Finding p and q

To generate the RSA key pair, prime numbers 8,087 (p) and 8,231 (q) were chosen and checked for primality through the Miller-Rabin primality test, passing 5 random test cases. 5 test cases were satisfactory which are utilised for higher 1024-bit numbers in checking primality.

1.2 Finding n

p and q were then multiplied together, producing n with the value of 66,564,097 and forming half of the public key. This is used in RSA as a modulus in the encryption and decryption, maintaining manageable value ranges and hiding the true value of the $base^{exponent}$ calculations. Due to n having a 26-bit length, it guarantees 25-bits can be encrypted and decrypted.

1.3 Finding phi

In order to find the values of e and d, phi must be calculated using the formula:

$$(p - 1) \times (q - 1)$$

Phi remains secret to ABSecure due to it being used for creating the private key. With ABSecure's p and q values, their phi is 66,547,780.

1.4 Finding e

e is a value greater than 0 and less than phi where the greatest common divisor between both are 1. Within phi, various possible e values exist, where within the first 100 values derived from phi, 37 of these met the requirement. A small e value of 9 was chosen, providing faster encryption over its larger alternatives, completing the public key.

1.5 Finding d

To find the value of the private key d, a value below phi is found meeting the formula:

$$d \times e \equiv 1 \pmod{phi}$$

By using e to find d, both values would be modular inverses where using e as an exponent modulo n can be reversed using d as an exponent modulo n and vice versa. With ABSecure's e and phi, the value 36,970,989 meets the requirement, forming the private key.

1.6. Key Pair Generated & Summary of Values

From the values summarised below, a public and private key pair are generated.

p: 8087

q: 8231

n: 66564097

phi: 66547780

e: 9

d: 36970989

Public key: (66564097, 9)

Private key: 36970989

1.7. RSA Encryption and Decryption

Due to e and d being modular inverses, the public key e can be used for encrypting the 576-bit keystream k, whilst the private key d can reverse this encryption, providing secure key exchange. For encryption and decryption to be successful, k as plaintext or ciphertext must be split into blocks; this is caused by the small value of n only allowing 25-bits to be safely encrypted and decrypted in a single calculation.

Each block of k is converted to its binary sequence, which is then converted to its decimal value forming the base of the exponent multiplication. The block then undergoes RSA encryption using the public key through the below formula and converted back to binary:

$$y = x^e \pmod{n}$$

Substituting for ABSecure's public key, it is the following:

$$y = block^9 \pmod{36970989}$$

For the decryption of a ciphertext block, the binary is converted into decimal and used as the base of the exponent multiplication. The block then undergoes decryption using the private key through the formula:

$$x = y^d \pmod{n}$$

Substituting for ABSecure's private key and modulus n, it is the following:

$$x = y^{36970989} \pmod{36970989}$$

The resulting decimal is converted to binary and padded to the length of the original block with its leading zeros. Each block is then concatenated together, reforming the key. Without padding, blocks may lose their leading zeros causing bits to be lost when concatenated back together and resulting in a different key.

1.8. Security

1.8.1. Mathematical Attack – Factorise p & q

Primes are used in the generation of RSA key pairs since it is challenging to factorise n and find the primes. Without the primes, the attacker is unable to determine phi and without phi, is unable to determine d. As a result, the attacker is unable to obtain the key and decrypt the ciphertext. With ABSecure's 26-bit n, its primes can be factorised within a few hours and will continue so until it is greater than 300-bits. As a result, the attacker can determine phi and the private key d. Achieving this, the attacker can decrypt the ciphertext of all keys during key exchange, obtaining the ability to decrypt all messages.

A solution to the factorisation problem is increasing the length of n to a value greater than 1024 bits, making it computationally expensive for the attacker to perform with no known fast factoring algorithm currently known.

1.8.2. Side-Channel Attack – Processor Timing to Detect Bit Values in Square-and-Multiply

This attack takes advantage of a processor and its timings when executing the square-and-multiply algorithm during encryption. In the algorithm, 0 valued bits perform only squaring operations, requiring less time to execute in comparison to 1 valued bits which perform a square and multiply. The attacker analyses how long it takes for the process to execute, determining that a bit is a 1 if it took longer and 0 if it was shorter.

A solution to this attack is performing a false multiplication after every 0-bit implementation. This causes the processor to execute 0-bit values in the same amount of time as 1-bit values due to both now performing squaring and multiplying; this eliminates the distinctions in execution times, reducing the ability for an attacker to determine the binary sequence of the key.

2. Hash Function

2.1 Properties of the Hash Function

2.1.1. Accepting any Length Input & Outputting a Short-Fixed Length

The hash function accepts messages of any bit-length, padding those which do not divide evenly into 192-bit multiples. For instance, a 1-bit message will pad 191-bits in order to create a 192-bit block that can be processed. Messages greater than the 192-bit input of the hash function for instance, X are split into 192-bit blocks that each undergo 45 rounds of hashing. Afterwards, the hashed block is XOR'd with the next block which then undergoes the hashing process. By using the outcomes of the previous block to influence the outcomes of the next, it results in every block of the plaintext having an impact in the output of the hash digest.

Once all blocks of a message have undergone hashing, the final digest in binary is converted to decimal and modulo 4,722,366,482,869,645,213,695 (highest 72-bit number). This reduces the hash digest to a maximum size of 72-bits, outputting a short 9-character hash digest.

2.1.2. One-Way Functionality

In section 2.2.1, the difficulties an attacker will face to determine the original message from its hash digest is discussed in detail, with the properties of one-way functionality and pre-image resistance sharing similar characteristics.

2.1.3. Sensitivity

With the use of exponent multiplication for substituting splits in the function, a single bit change in a message will cause different bases and exponents to be multiplied together. These differences will produce a large change in the decimal result and thus, binary sequence of the group when concatenated together and the next exponent multiplication between groups. The group with the single bit change will have a different value as its base and exponent, drastically changing the result it produces with another group. The avalanche effect continues, changing the result of the bit shift and continuing to impact the inputs of the next 44 rounds of base exponent multiplications.

At the end of a block's hashing, the bit change results in a different hash digest, causing the XOR with the next block to also differ, manipulating the outcome of the next block and all blocks after.

2.1.4. Efficiency

The hash function was implemented in python producing a hash digest of the message X (performed on a MacBook Pro, M1 pro chip, 16gb). Across 10 trials, it was discovered the hash digest can be produced in ~30 milliseconds. The bank may perform 2.2 million transactions per day (~26 transactions per second), with the hash function producing ~33 hash digests per second, proving its efficiency with hashing transactions (Reserve Bank of Australia, 2022).

2.2. Security Properties of the Hash Function

2.2.1 Pre-Image Resistance

Given a hash digest, it is extremely difficult for an attacker to determine the message X.

Through compression of the 192-bit value to 72-bits through a modulus, the true size of the 192-bit value is obscured with multiple values resulting in the same modular 72-bits. In addition, within this obscured 192-bit block, many exponent multiplications occurred. The attacker would have the final 192-bit block and the modulus however, not the value of the groups before exponent multiplication. The attacker would need to find multiple bases and exponents to find the group values. If found, the attacker would then have to determine the base and exponents of each split for the 4 groups, with 12 splits per round. This only notes a single hashing round, with 44 additional rounds of base exponent calculations to reverse. If a message requires 6 blocks to be iterated through to produce a hash digest, 8,640 exponent multiplications are to be reversed (6 blocks x (45 rounds x 16 multiplications)).

Despite this, shortcomings exist. Since the base and exponents for the substitution of a split is created from the same binary sequence, an attacker only needs to test 2^{16} possible combinations, since one of those sequences contains both the base and exponent value. A solution to this is to use another split within the group as the exponent; this will cause the substitution of a split to rely on two separate splits, requiring the attacker to test 2^{16} possible exponents for each 2^{16} possible bases. As a result, finding all bases and exponents becomes more computationally expensive and time-consuming.

2.2.2. Collision Resistance

For a collision to be found, 2^{36} (68,719,476,736) messages will need to be checked. With the bank possibly performing 22 million transactions daily, the bank will produce unique hashes for 31,236 days (86.7 years) (Reserve Bank of Australia, 2022). Despite this, collisions will occur earlier. By reducing 192-bits into 72-bit values through modulus, multiple 192-bit values will produce the same 72-bit value causing a collision to occur. For instance, the decimal values 1 and 4,722,366,482,869,645,213,696 will produce the same 72-bit value 1, resulting in the same hash digest with different messages. This will cause our message X to collide with other random messages.

This can be resolved by increasing the output of the hash digests to 192-bits, eliminating the need for compression and preserving the level of mixing performed within the blocks. In addition, it will increase the length of the hash, producing a greater number of unique hashes (2^{96} would need to be checked for collisions).

2.2.3. Second Pre-Image Resistance

Although second pre-image resistance differs from collision resistance, its discussion remains the same. ABSecure will have 86.7 years of unique hash digests however, will still have second pre-image resistance be impacted by the way compression is implemented. The use of a modulus for compressing a 192-bit decimal into a 72-bit decimal cause two randomly generated messages to produce the same value.

The same solution would be applied, increasing the output to 192-bits to produce longer hash digests, provide a greater number of unique hashes available and eliminate the need for a compression algorithm.

3. RSA digital Signature

To validate the integrity of the message and authenticity of ABSecure, RSA digital signature's will be implemented. This involves ABSecure calculating a signature from their hash digest using the private key, sending it and the original message to the receiving bank. The receiving bank then decrypts the signature using ABSecure's public key, checking if the original message and the decrypted signature match. For signing messages, its 72-bit hash digest equivalent will be used, allowing the plaintext to remain secure and to reduce the resources required for calculating signatures which large plaintexts cause.

3.1. Use of Verified Signatures

A verified signature provides message authentication to the receiving bank, showing the message has not been manipulated and is genuine. In addition, it provides authenticity of ABSecure as they are the only entity which know the private key therefore, are the only entity which can produce signatures that are decrypted by their e. ABSecure will not be able to breach non-repudiation since they cannot deny the sending of a message which is tied to their private key.

3.2. Reasoning for RSA

RSA was chosen for producing digital signatures due to its speed in verifying signatures over DSA. The bank will perform message verification multiple times for instance, when individuals request their transactions or the bank investigating the source of transactions. Where a message would be signed only once, it would be verified multiple times. Furthermore, RSA is also used for secure key exchange as seen in section 1, meaning a single public key provides multiple functions, reducing the management of multiple key pairs.

3.3. Key Pair Generation

Primes 1,000,000,000,163 (p) and 1,000,000,006,793 (q) were chosen, passing 5 test cases of the Miller-Rabin primality test. These primes created an n of 1,000,000,006,956,000,001,107,259. This is an 80-bit n, allowing hash digests to be signed within a single block. To find e and d, phi was found with the value 1,000,000,006,954,000,001,100,304. An e value of 9 was chosen as it satisfied the requirement discussed in section 1.4 and finally, the value of d was determined as 555,555,559,418,888,889,500,169 meeting the requirement discussed in section 1.5. A small e was chosen to allow verifying signatures to be computationally easy, reducing the amount of time it would take for receiving banks to verify ABSecure's messages. Although this causes a large d value and longer signing times, message verification usually occurs multiple times whilst signing once therefore, verification speed is prioritised.

ABSecure now generates their public and private key, releasing the public key to receiving banks. A summary of the values generated and the key pairs are seen below.

p: 1000000000163
q: 1000000006793
n: 1000000006956000001107259
phi: 1000000006954000001100304
e: 9
d: 555555559418888889500169

Public key: (1000000006956000001107259, 9)
Private key: 555555559418888889500169

3.4. Calculating a Signature

To calculate the signature of a hash digest, it is converted to its decimal equivalent and used as the base in the formula:

$$s = x^d \pmod n$$

Substituting for ABSecure's hash digest of X and their private key, it is the following:

$$h(x) = ;OaP@*x6j$$
$$h(x)_{10} = 16918268217146378650$$

$$s = 16918268217146378650^{555555559418888889500169} \pmod{100000000695600000 1107259}$$
$$s = 307191254441189051314448$$

The hash and its signature are then sent to the receiving bank in the format:

$$(h(x), s)$$
$$(;OaP@*x6j , 307191254441189051314448)$$

3.5. Verifying a Signature

The receiving bank will use ABSecure's public key e to reverse the signature using the formula:

$$z = s^e \pmod n$$

The value of z is converted into binary and compared with the binary of the hash digest. If both have the same binary sequence, then verification is successful and the message is authenticated.

Substituting for ABSecure's hash digest of X and their private key, it is the following:

$$z = 307191254441189051314448^9 \pmod{100000000695600000 1107259}$$
$$z = 15395350994564831191 63_{10}$$
$$z = 0101001101110101010101110110100000100110001011010101101001010000001110 11_2$$

With the signature reversed the binary of z is compared with the binary of the message X's hash digest, revealing both are the same and proving the message's authenticity.

$$h(x) = 01010011011101010101011101101000001001100010110101011010010100000011101 1$$

$$z = 01010011011101010101011101101000001001100010110101011010010100000011101 1$$

4. Summary of Cryptosystem

4.1. Cryptosystem

This hybrid cryptosystem utilises symmetric and asymmetric cryptography, providing ABSecure the ability to encrypt messages and securely exchange keys for decryption, whilst also providing receiving banks the ability to authenticate these messages and the entity providing them.

4.1.1. Steps to Securely Exchange and Secure the Message X

To transmit and secure the message X, ABSecure can follow the steps below.

1. Produce a hash digest of X.
2. Create keystream k the length of X in binary through stream cipher.
3. Encrypt X with k.
4. Encrypt k using the public key of receiving bank to prepare key exchange, allowing only the receiving bank to obtain k.
5. Produce a digital signature of hash digest using ABSecure's private key.
6. Send the ciphertext, signature and exchange k, providing the receiving bank all information to verify the message and decrypt the ciphertext.

4.1.2. Use of Symmetric and Asymmetric Cryptography in Hybrid Cryptosystem

Within this hybrid cryptosystem, symmetric cryptography provides fast message encryption through its use of XOR operations between plaintext and keys, whilst asymmetric cryptography allows this key to be exchanged securely, in addition to providing message and entity authentication through digital signatures.

If symmetric keys were used for the key exchange process, both banks would have the ability to produce encrypted messages. This can lead to breaches in non-repudiation since a bank can create plaintext and deny sending it, placing responsibility on the other bank. In addition, it would provide the attacker multiple entities to obtain the key from, increasing the possibly of obtaining it. Finally, both banks would need a secure channel for deciding and sharing this key to stop attackers from intercepting it.

Asymmetric cryptography is advantageous in these situations, utilising keypairs which ABSecure can produce alone, eliminating the need for shared key discussions with the other bank. In addition, ABSecure's public key can be used by multiple banks rather than requiring a different shared key between each bank, reducing the number of keys to be managed. Furthermore, the keypairs solve the issue of non-repudiation through digital signatures which only ABSecure can create with their private key; ABSecure can no longer deny sending messages as it has been signed by their private key. Finally, the signatures provide message integrity as invalid signatures indicate a manipulated message during transmission, in addition to proving the sender of the message is authentic. The use of hash digests in digital signatures is also to be noted, allowing messages to be validated in signatures without providing the plaintext.

4.2. Cryptanalysis of the Hybrid Cryptosystem

A large weakness of this hybrid cryptosystem is the small 40-bit primes used in the RSA key pair generation. As discussed in section 1.8.1, an 80-bit n can be factorised within a few hours. This would cause severe consequences for ABSecure, allowing the attacker to determine the private key and therefore, be able to decrypt all ciphertexts sent to ABSecure. This means regardless of the security provided by encryption algorithms, it is bypassed.

Another weakness that can be exploited is the square-and-multiply method. As discussed in section 1.8.2, the time taken to execute 0 and 1 valued bits by the processor can be used to find the value of the key before encryption. Due to the cryptosystem not implementing numbers used only once (nonce) to randomise the keystreams, all iterations of the keystream will produce the same output (assuming initialising bit values for linear feedback shift registers remain the same). As a result, the attacker obtains the key used for decrypting all ciphertexts.

With access to the plaintexts, the attacker can manipulate messages to increase the value of a transaction or the address of the sender and receiver.

5. Personal Reflection

Through undertaking the portfolio tasks, I have a greater understanding on asymmetric encryption, a category of cryptography excelling in fast encryption but held back by its use of shared keys, in addition to asymmetric encryption which uses key pairs for providing message authentication and the secure key exchange of asymmetric keys. This assignment has taught me that by utilising both, a hybrid cryptosystem which takes advantage of these strengths can be produced. Furthermore, by having the opportunity to design my own ciphers and hash functions, I have achieved a greater understanding of the techniques for instance, shifting and mixing which can be used to secure plaintext and how they should be applied. This level of understanding would not have been achieved if I was unable to apply my learnings.

Despite this, I found creating a hash function which satisfied all general and security properties whilst also creating small, unique outputs challenging. I struggled in combining the techniques learnt to ensure a creative and unique hashing algorithm was created; this led to myself spending large quantities of time staring at the screen with no steps forward. I believe from this time spent however, it led to myself implementing exponent multiplication which became a core component in producing unique and one-way hash digests in my function. Regardless I am happy with the result I achieved, believing I created a complex hash function which met all properties.

5.1. Misconceptions Corrected

Before commencing the unit, I had no prior understanding of cryptology and its complexity. Only once I began attempting to create a secure algorithm did I realise how impressive current standards like AES and redundant algorithms for instance, DES are. The portfolio tasks made me appreciate cryptography and its algorithms since without these, the privacy of online information would not exist.

Before the unit I also had the misconception that the current algorithms we utilise have no known weaknesses due to a lack of information on them. I was proven wrong in the 4th week

of this unit where I learnt in detail the structure of AES and its various techniques for performing encryption. To learn that a well-researched algorithm has no known weaknesses and is used as a government standard for the United States was eye-opening, realising the best ciphers are those which are well-known and have shown to contain no weaknesses.

5.1.2. My Perspective on Cryptology

From attempting to create a secure hybrid cryptosystem, my perspective on cryptology is that it is easy to pick up but hard to master. With the knowledge gained throughout the unit I successfully created a complex cryptosystem in design. Although a positive, this complex system has large weaknesses which can be easily exploited and bypass most of the cryptosystem. To create a cryptosystem which becomes industry standard takes time, experience and understanding of complex concepts and mathematics which I do not possess. This unit only scratches the surface of cryptology and although I do not have the personal attributes for pursuing it as a career, it has become a personal interest of mine where I want to continue learning about cryptology. From this interest, I want to learn how to improve my cryptosystem to create a product I am confident in expressing in my career development.

6. References

Reserve Bank of Australia. (2022). *Reverse Bank of Australia Annual Report – 2022*. https://www.rba.gov.au/publications/annual-reports/rba/2022/banking-and-payment-services.html