RSA THEORY UNDERSTANDING

RSA (Rivest–Shamir–Adleman) is one of the first public-key cryptosystems and it’s an algorithm used by computers to encrypt and decrypt messages. RSA is widely used for securing sensitive data, particularly when being sent over an insecure network such as the Internet.

It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone. The other key must be kept private. These two keys can help to encrypt the message; the opposite key from the one used to encrypt a message is used to decrypt it. Because of this reason, RSA has become the most widely used asymmetric algorithm. It provides a method of assuring the confidentiality, integrity, authenticity and non-reputability of electronic communications and data storage.

RSA security cines from the difficulty of factoring large integers that are the product of 2 large prime numbers. It’s easy to multiply the two numbers but to know what original prime numbers are is a hard task for humans and for computers as well.

How these 2 keys are generated is the complex part of RSA. We get 2 large prime numbers, p and q that are generated by using the Rabin-Miller primality test algorithm. We also need a modulus n that is calculated by multiplying p and q. This number is used by both the public and private keys and provides the link between them. Its length, usually expressed in bits, is called the key length. The public key consists of the modulus n, and a public exponent, e, which is normally set at 65537, as it's a prime number that is not too large. The private key consists of the modulus n and the private exponent d, which is calculated using the Extended Euclidean algorithm to find the multiplicative inverse with respect to the totient of n.

EXAMPLE:

Matt generates her RSA keys by selecting two primes: p = 11 and q = 13. To find n we say n = p x q = 143. To find the totient of n we say: ϕ(n) = (p − 1) x (q − 1) = 120. He decides to use 7 as her RSA public key e and calculates his RSA private key using the extended Euclidean Algorithm which gives him 103

Mike wants to send Matt and encrypted message M so he obtains his RSA public key (n, e) which is (147, 7). Mike’s plaintext message is just the number 9 and is encrypted like this:

When Matt receives Mike’s message, he decrypts it by using his RSA private key (d, n):

To use RSA keys to digitally sign a message, Matt would create a hash or message digest of his message to Mike, encrypt the hash value with his RSA private key and add it to the message. Mike can then verify that the message has been sent by Matt and has not been altered by decrypting the hash value with his public key. If this value matches the hash of the original message, then only Matt could have sent it (authentication) and the message is exactly as he wrote it (integrity). Matt could, of course, encrypt his message with Mike’s RSA public key (confidentiality) before sending it to Mike. A digital certificate contains information that identifies the certificate's owner and also contains the owner's public key. Certificates are signed by the certificate authority that issues them and can simplify the process of obtaining public keys and verifying the owner.

the security of RSA relies on the computational difficulty of factoring large integers. As computing power increases and more efficient factoring algorithms are discovered, the ability to factor larger and larger numbers also increases. Encryption strength is directly tied to key size, Normally, RSA keys are 1024-2048 bits long, but experts believe that 1024-bit keys could be broken in the near future, which is why government and industry are moving to a minimum key length of 2048-bits.

# MODERN SOFTWARE APPLICATIONS

## cAESAR CIPHER PROOF OF INSECURITY

Caesar’s cipher is one of the oldest, simplest, and most well-known cipher’s in the world. This cipher was named after Julius Caesar, who use it to send private correspondence of military significance. This cipher I a type of substitution cipher where each letter in the plaintext is replaced by a letter that’s been fixed by a number of a letter in the alphabet. For example, if I have the phrase “ABCDE” and I decide to shift it with 3 to the Left, the phrase will be “XYZAB”:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Y | Z | A | B | C | D | E | F | G |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T | U | V | W | X | Y | Z | A | B | C | D | E | F |

The Caesar cipher can be broken easily. There are two situations that need to be considered:

1. If the attacker knows or guesses that a substitution cipher has been used (not specifically a Caesar cipher)
2. The attacker know that Caesar cipher is been used but doesn’t know the shift and direction

For the first situation, the cipher can be broken using the same techniques for a general simple substitution cipher, like a pattern of words. An attacker can easily notice the regularity in the solution (the plaintext has remained intact) and they discover patterns and and deduce that a Caesar cipher has been used.

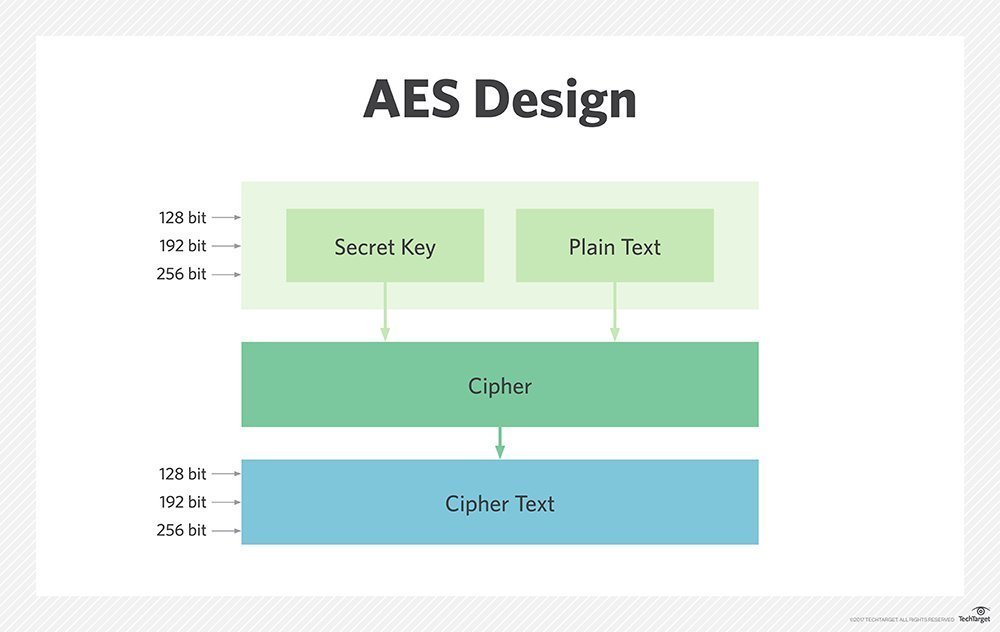
For the second situation, breaking it will be really straightforward. There are only 26 possible shifts (in the English language), testing all the possible solutions. Also, the mathematical formula can come in handy to hack into the cipher:

Where **x** is the letter, **n** is how many letters is the original letter going to be shifted. To find the encrypted text (or ciphertext) it will really similar to the one to create the ciphertext but instead of adding we will subtract:

AES

AES states for Advanced Encryption Standard. AES is a subset of the Rijndael cipher. It was first adapted by the US government and now it’s used worldwide. This cipher is responsible for all the large amount of the information security that we enjoy daily.

AES is based on a design principle known as “Substitution-permutation network” which is a series of linked mathematical operations that are used in block cipher algorithms. AES has a fixed block with a size of 128, and key sizes of 128, 102, or 256 bits.



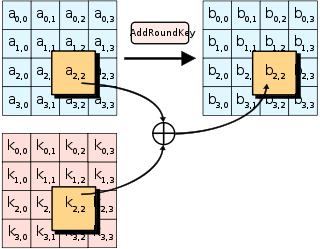
The AEs encryption algorithm defines a number of transformations that are to be performed on data stored in an array. The first step of the cipher is to put the data into an array where the cipher transformations are repeated over a number of encryption rounds. The number of rounds is determined by the length of the key, with 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys.

The steps for this algorithm are:

1. Have your key and the phrase you want to encrypt
2. Initial round
   1. AddRoundKey - Each byte of the state\* is combined with a block of the round key using bitwise xor
3. Second Round
   1. SubBytes – A non-linear substitution step where each byte is replaced with another according a look up table (Rijndael S-Box\*)
   2. ShiftRows – A transposition step where the last three rows of the table are shifted cyclically a certain number of steps
   3. MixColumns – A mixing operation which operate on the columns of the state combining the four bytes in each column
   4. AddRoundKey
4. Final round
   1. subBytes
   2. ShiftRows
   3. AddRoundKey

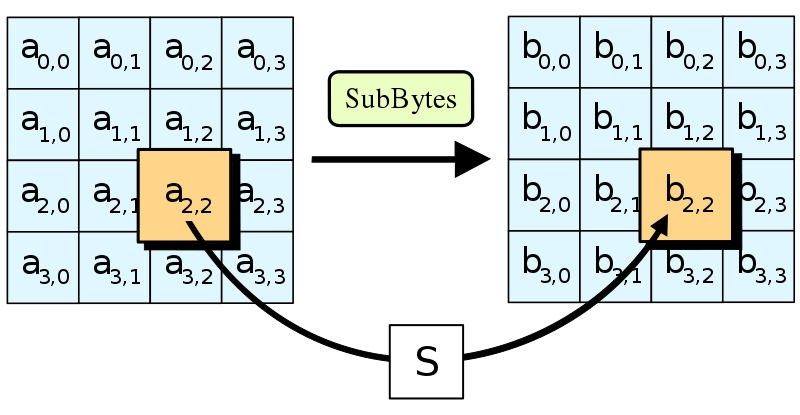
**AddRoundKey**

In this step, the subkey is combined with the state. For each round, a subkey is derived from the main key using using Rijndael's key schedule\*; each subkey is the same size as the state. The subkey is added by combining each byte of the state with the corresponding byte of the subkey using bitwise XOR



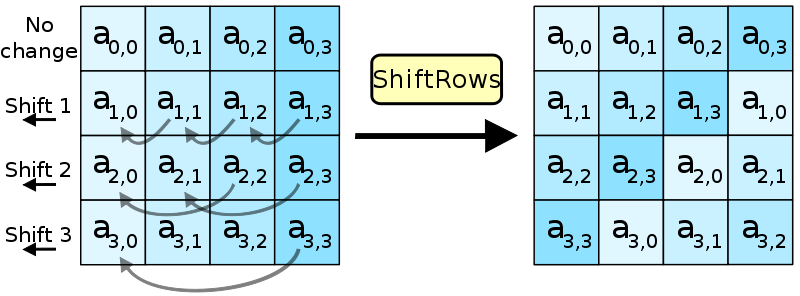
**SubBytes**

Each byte aij in the state matrix is replace with a SubByte S(aij) using a using an 8-bit substitution box, the Rijndael S-box. This operation provides the non-linearity in the cipher.



**ShiftRows**

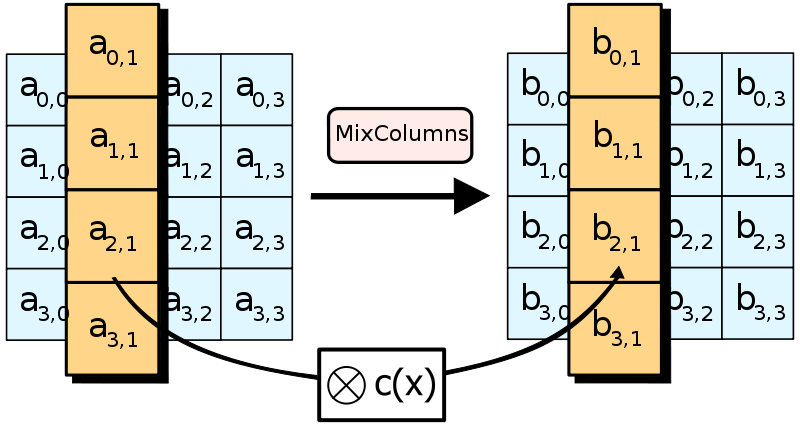
This step operates on the rows of the state; it cyclically shifts the bytes in each row by a certain offset. For AES, the first row is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. For blocks of sizes 128 bits and 192 bits, the shifting pattern is the same. For a 256-bit block, the first row is unchanged and the shifting for the second, third and fourth row is 1 byte, 3 bytes and 4 bytes respectively—this change only applies for the Rijndael cipher when used with a 256-bit block, as AES does not use 256-bit blocks. The importance of this step is to avoid the columns being encrypted independently, in which case AES degenerates into four independent block ciphers.



**MixColumns**

In this step, In the MixColumns step, the four bytes of each column of the state are combined using an invertible linear transformation. The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Together with ShiftRows, MixColumns provides diffusion in the cipher.

During this operation, each column is transformed using a fixed matrix (matrix left-multiplied by column gives new value of column in the state):



TERMS FROM STEPS:

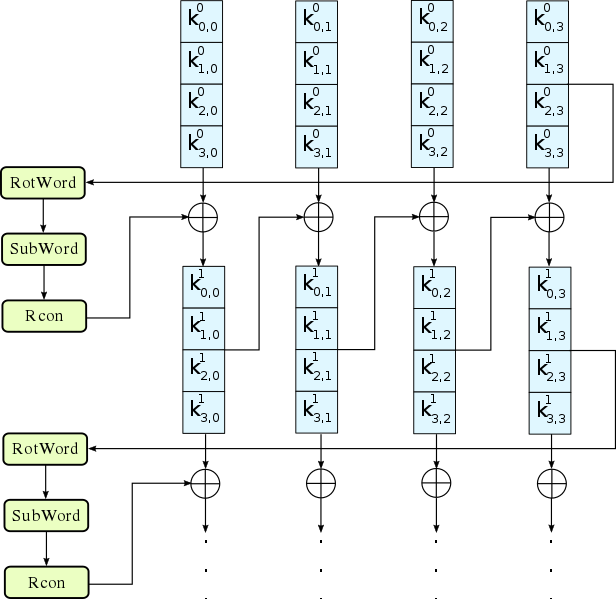
*State:* A 4 x 4 column-major order matrix of bytes

*Rijndael's key schedule:* a key schedule to expand a short key into a number of separate round keys. It uses a rotate operation that takes a 32-bit word in hexadecimal and it rotates its bits to the left such that the top bits “wrap around” and become the low eight bits of the result.

Since the key schedule for 128-bit, 192-bit, and 256-bit encryption are very similar, with only some constants changed, the following keysize constants are defined here:

* n has a value of 16 for 128-bit keys, 24 for 192-bit keys, and 32 for 256-bit keys
* b has a value of 176 for 128-bit keys, 208 for 192-bit keys, and 240 for 256-bit keys (with 128-bit blocks as in AES, it is correspondingly larger for variants of Rijndael with larger block sizes).

1. The first n bytes of the expanded key are simply the encryption key.
2. The rcon iteration value i is set to 1
3. Until we have b bytes of expanded key, we do the following to generate n more bytes of expanded key:
   * 1. We do the following to create 4 bytes of expanded key:
   1. We create a 4-byte temporary variable, t
   2. We assign the value of the previous four bytes in the expanded key to t
   3. We perform the key schedule core (see above) on t, with i as the rcon iteration value
   4. We increment i by 1
   5. We exclusive-OR t with the four-byte block n bytes before the new expanded key. This becomes the next 4 bytes in the expanded key
      1. We then do the following three times to create the next twelve bytes of expanded key:
   6. We assign the value of the previous 4 bytes in the expanded key to t
   7. We exclusive-OR t with the four-byte block n bytes before the new expanded key. This becomes the next 4 bytes in the expanded key
      1. If we are processing a 256-bit key, we do the following to generate the next 4 bytes of expanded key:
   8. We assign the value of the previous 4 bytes in the expanded key to t
   9. We run each of the 4 bytes in t through Rijndael's S-box
   10. We exclusive-OR t with the 4-byte block n bytes before the new expanded key. This becomes the next 4 bytes in the expanded key.
       1. If we are processing a 128-bit key, we do not perform the following steps. If we are processing a 192-bit key, we run the following steps twice. If we are processing a 256-bit key, we run the following steps three times:
   11. We assign the value of the previous 4 bytes in the expanded key to t
   12. We exclusive-OR t with the four-byte block n bytes before the new expanded key. This becomes the next 4 bytes in the expanded key



CRYPTANALYSIS

