Introduction to Cryptography

I. Introduction

Mục đích của phần này là giới thiệu cho người dùng các khái niệm mật mã học cơ bản như:

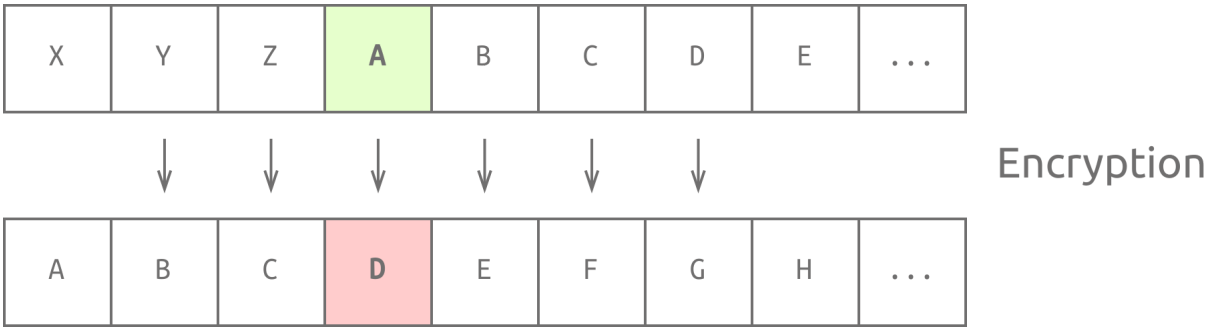
Mã hóa đối xứng, chẳng hạn như AES

Mã hóa không đối xứng, chẳng hạn như RSA

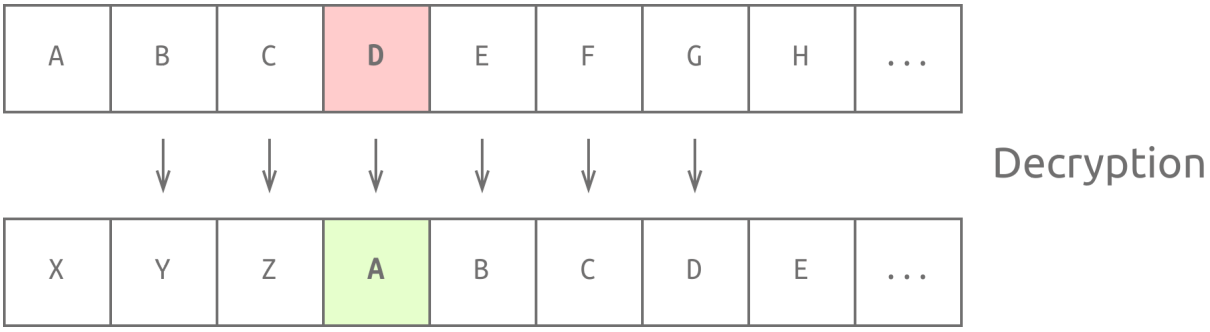
Trao đổi khóa Diffie-Hellman Băm PKI

Giả sử bạn muốn gửi một tin nhắn mà không ai có thể hiểu được ngoại trừ người nhận dự định. Bạn sẽ làm thế nào?

Một trong những mã đơn giản nhất là mã Caesar, được sử dụng hơn 2000 năm trước. Mã Caesar dịch chuyển chữ cái theo một số lượng cố định sang trái hoặc sang phải. Hãy xem xét trường hợp dịch chuyển 3 sang phải để mã hóa, như được hiển thị trong hình dưới đây.



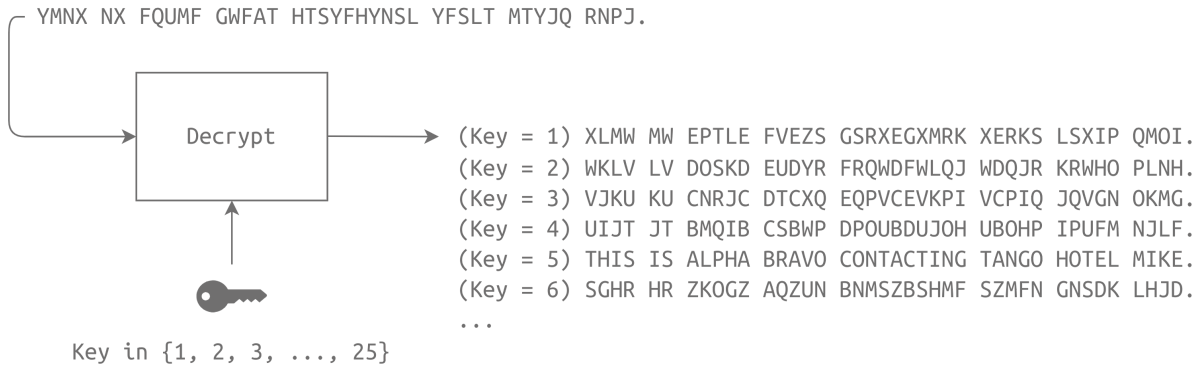
Người nhận cần biết rằng văn bản đã được dịch chuyển sang phải 3 đơn vị để khôi phục lại thông điệp ban đầu.



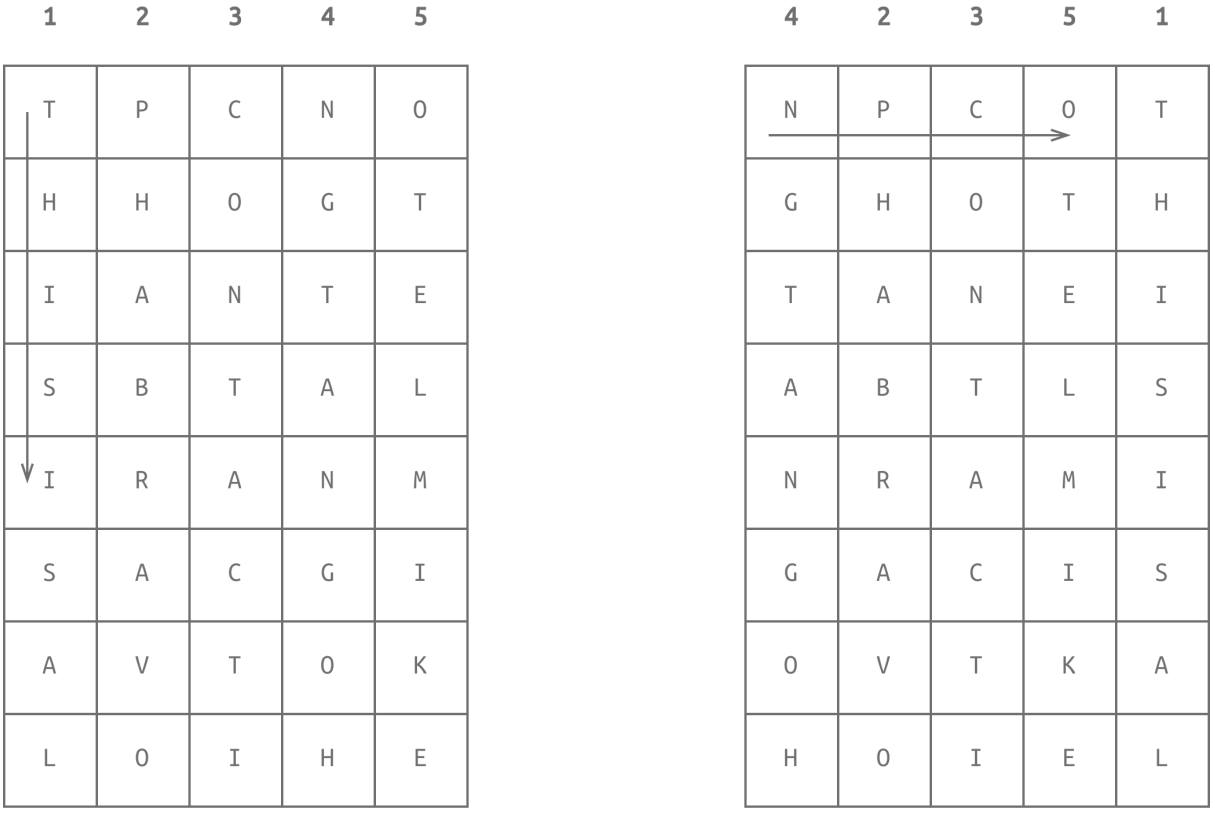
Sử dụng cùng một khóa để mã hóa “TRY HACK ME”, chúng ta sẽ được “WUB KDFN PH”.

Mã Caesar mà chúng tôi đã mô tả ở trên có thể sử dụng một khóa từ 1 đến 25. Với một khóa bằng 1, mỗi chữ cái được dịch chuyển một vị trí, trong đó A trở thành B và Z trở thành A. Với một khóa bằng 25, mỗi chữ cái được dịch chuyển 25 vị trí, trong đó A trở thành Z và B trở thành A. Một khóa bằng 0 có nghĩa là không thay đổi; hơn nữa, một khóa bằng 26 cũng sẽ dẫn đến không thay đổi vì nó sẽ dẫn đến một vòng quay đầy đủ. Do đó, chúng tôi kết luận rằng Mã Caesar có một không gian khóa gồm 25; có 25 khóa khác nhau mà người dùng có thể chọn.

Hãy xem xét trường hợp bạn đã chặn được một tin nhắn được mã hóa bằng Mã Caesar: “YMNX NX FQUMF GWFAT HTSYFHYNSL YFSLT MTYJQ RNPJ”. Chúng tôi được yêu cầu giải mã nó mà không có kiến thức về khóa. Chúng tôi có thể thử bằng cách sử dụng vét cạn, tức là chúng tôi có thể thử tất cả các khóa có thể và xem khóa nào là hợp lý nhất. Trong hình dưới đây, chúng tôi nhận thấy rằng khóa là 5 là hợp lý nhất, “THIS IS ALPHA BRAVO CONTACTING TANGO HOTEL MIKE.”



Mật mã Caesar được coi là một loại mật mã thay thế, trong đó mỗi chữ cái của bảng chữ cái được thay thế bằng một chữ cái khác. Mật mã chuyển vị là một loại mật mã khác, mã hóa thông điệp bằng cách thay đổi thứ tự các chữ cái. Hãy xem xét một mật mã chuyển vị đơn giản trong hình dưới đây. Chúng ta bắt đầu với thông điệp “THIS IS ALPHA BRAVO CONTACTING TANGO HOTEL MIKE” và khóa 42351. Sau khi viết các chữ cái của thông điệp của chúng ta bằng cách điền một cột sau đó là cột khác, chúng ta sắp xếp lại các cột dựa trên khóa và sau đó đọc các hàng. Nói cách khác, chúng ta viết theo cột và đọc theo hàng. Chú ý rằng chúng tôi đã bỏ qua tất cả các khoảng trắng trong văn bản gốc trong ví dụ này. Kết quả mã hóa “NPCOTGHOTH…” được đọc một hàng sau đó là hàng khác. Nói cách khác, mật mã chuyển vị đơn giản chỉ sắp xếp lại thứ tự các chữ cái, khác với mật mã thay thế, thay thế các chữ cái mà không thay đổi thứ tự của chúng.



Nhiệm vụ này giới thiệu các mật mã thay thế và chuyển vị đơn giản và áp dụng chúng vào các thông điệp được tạo từ các ký tự chữ cái. Để một thuật toán mã hóa được coi là an toàn, nó phải không thể khôi phục được thông điệp ban đầu, tức là văn bản gốc. (Trong thuật ngữ toán học, chúng ta cần một vấn đề khó, tức là một vấn đề không thể giải quyết trong thời gian đa thức. Một vấn đề mà chúng ta có thể giải quyết trong thời gian đa thức là một vấn đề có thể giải quyết được ngay cả đối với đầu vào lớn, mặc dù máy tính có thể mất một thời gian khá lâu để hoàn thành.)

Nếu thông điệp được mã hóa có thể bị phá vỡ trong một tuần, thì mã hóa được sử dụng sẽ được coi là không an toàn. Tuy nhiên, nếu thông điệp được mã hóa có thể bị phá vỡ trong 1 triệu năm, thì mã hóa sẽ được coi là an toàn trong thực tế.

Hãy xem xét mật mã thay thế đơn giản, trong đó mỗi chữ cái của bảng chữ cái được thay thế bằng một chữ cái khác. Ví dụ, trong tiếng Anh, bạn sẽ ánh xạ “a” đến một trong 26 chữ cái tiếng Anh, sau đó bạn sẽ ánh xạ “b” đến một trong 25 chữ cái tiếng Anh còn lại, và sau đó ánh xạ “c” đến một trong 24 chữ cái tiếng Anh còn lại, và cứ như vậy.

Ví dụ, chúng ta có thể chọn các chữ cái trong bảng chữ cái “abcdefghijklmnopqrstuvwxyz” để được ánh xạ đến “xpatvrzyjhecsdikbfwunqgmol” tương ứng. Nói cách khác, “a” trở thành “x”, “b” trở thành “p”, và cứ như vậy. Người nhận cần phải biết khóa “xpatvrzyjhecsdikbfwunqgmol” để giải mã các tin nhắn được mã hóa thành công.

Thuật toán này có vẻ rất an toàn, đặc biệt là khi thử tất cả các khóa có thể không khả thi. Tuy nhiên, các kỹ thuật khác nhau có thể được sử dụng để phá vỡ một văn bản được mã hóa bằng thuật toán mã hóa như vậy. Một điểm yếu của thuật toán này là tần số chữ cái. Trong văn bản tiếng Anh, các chữ cái phổ biến nhất là ‘e’, ‘t’ và ‘a’, vì chúng xuất hiện với tần suất lần lượt là 13%, 9,1% và 8,2%. Hơn nữa, trong văn bản tiếng Anh, các chữ cái đầu tiên phổ biến nhất là ‘t’, ‘a’ và ‘o’, vì chúng xuất hiện với tần suất lần lượt là 16%, 11,7% và 7,6%. Thêm vào đó, hầu hết các từ trong văn bản đều là từ điển, và bạn sẽ có thể phá vỡ một văn bản được mã hóa với mật mã thay thế chữ cái trong thời gian ngắn.

Chúng ta không cần thực sự sử dụng khóa mã hóa để giải mã văn bản được nhận, “Uyv sxd gyi siqvw x sinduxjd pvzjdw po axffojdz xgxo wsxcc wuidvw.” Như được hiển thị trong hình dưới đây, sử dụng một trang web như quipqiup, chúng ta sẽ mất một khoảnh khắc để khám phá ra rằng văn bản gốc là “The man who moves a mountain begins by carrying away small stones.” Ví dụ này rõ ràng cho thấy thuật toán này đã bị phá vỡ và không nên được sử dụng cho việc truyền thông tin mật.

---

quipqiup beta3

quipqiup is a fast and automated cryptogram solver by Edwin Olson. It can solve simple substitution ciphers often found in newspapers, including puzzles like cryptoquips (in which word boundaries are preserved) and patristocrats (inwhi chwor dboun darie saren t).

Puzzle:

"Uyv sxd gyi siqvw x sinduxjd pvzjdw po axffojdz xgxo wsxcc wuidvw."

Clues: For example G=R QVW=THE

Solve

0	-1.916	"The man who moves a mountain begins by carrying away small stones."
1	-2.679	"The man who moves a mountain pekins pr jaffrink awar small stones."
2	-2.753	"The man who moves a mountain jedins jr kaggrind awar small stones."

Answer the questions below

You have received the following encrypted message:

"Xjnvw lc sluxjmw jsqm wjpmcqbg jg wqcxqmnvw; xjzjmmjd lc wjpm sluxjmw jsqm bqccqm zqy." Zlwvzjxj Zpcvcol

You can guess that it is a quote. Who said it?

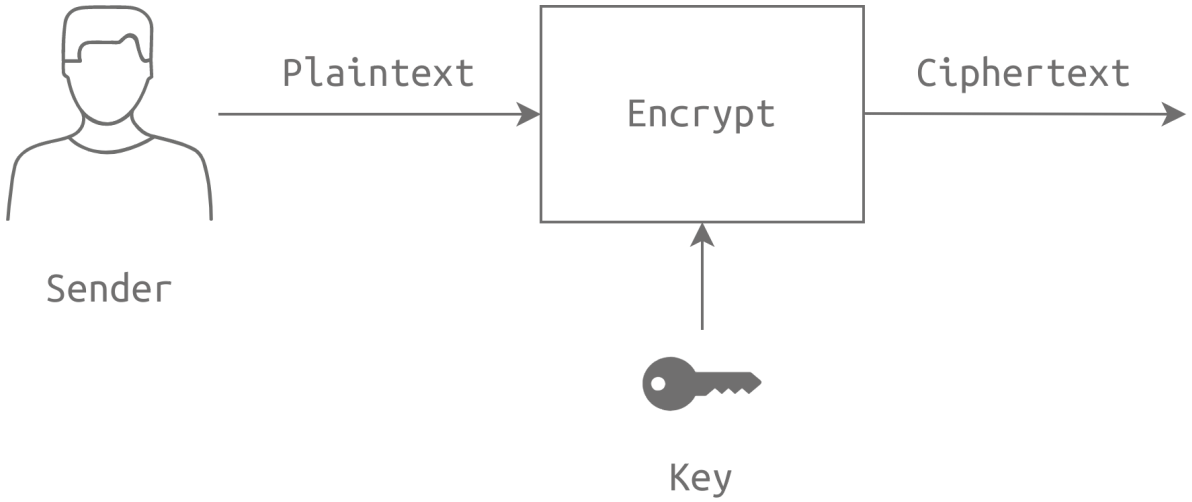
Miyamoto Musashi | Correct Answer | Hint

II. Symmetric Encryption

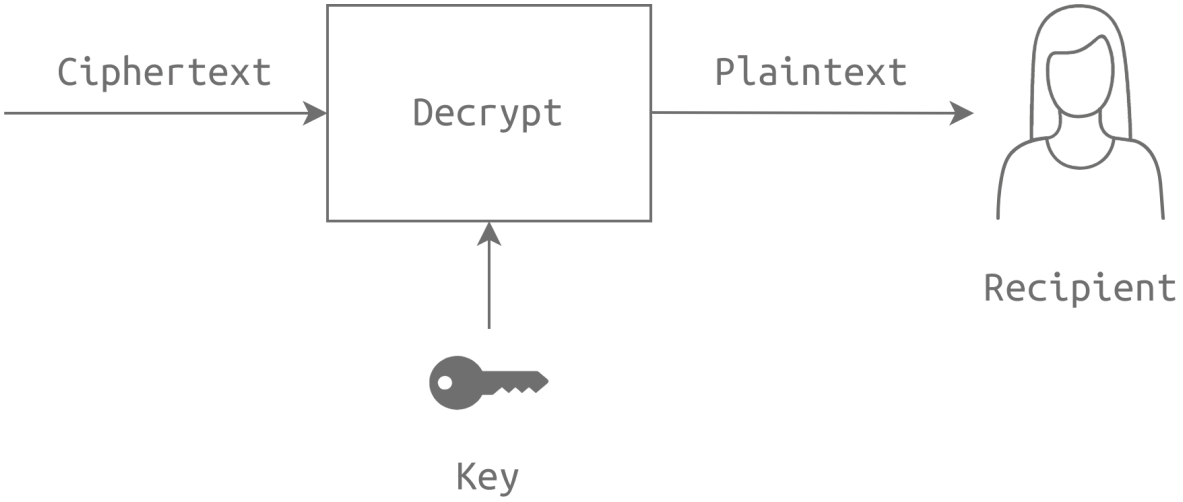
Hãy xem xét một số thuật ngữ:

Thuật toán mật mã học hoặc Cipher: Thuật toán này xác định quá trình mã hóa và giải mã. Khóa: Thuật toán mật mã học cần một khóa để chuyển đổi văn bản gốc thành văn bản được mã hóa và ngược lại. Văn bản gốc là thông điệp ban đầu mà chúng ta muốn mã hóa Văn bản mã hóa là thông điệp ở dạng được mã hóa Một thuật toán mã hóa đối xứng sử dụng cùng một khóa cho cả quá trình mã hóa và giải mã. Do đó, các bên liên lạc cần đồng ý về một khóa bí mật trước khi có thể trao đổi bất kỳ tin nhắn nào.

Trong hình dưới đây, người gửi cung cấp quá trình mã hóa với văn bản gốc và khóa để nhận được văn bản được mã hóa. Văn bản mã hóa thường được gửi qua một kênh truyền thông nào đó.



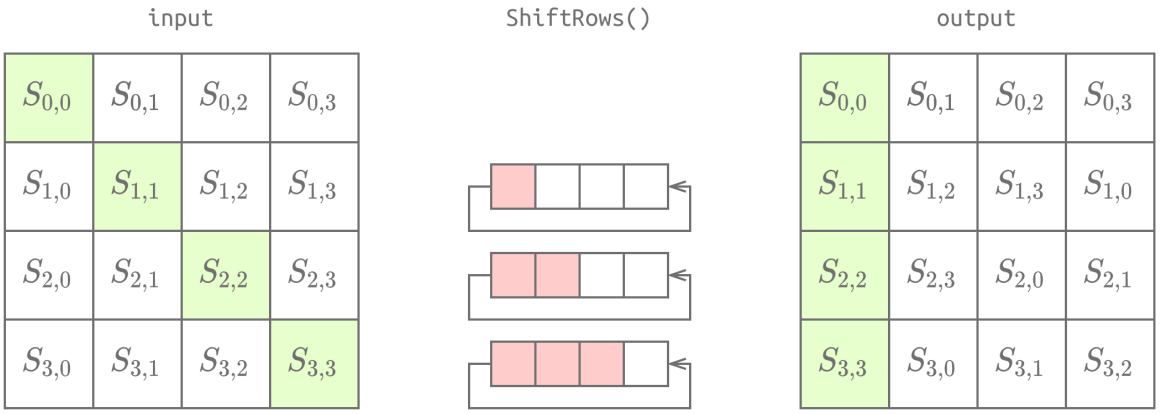
Ở phía người nhận, họ cung cấp quá trình giải mã với cùng một khóa được sử dụng bởi người gửi để khôi phục lại văn bản gốc từ văn bản được mã hóa nhận được. Nếu không có kiến thức về khóa, người nhận sẽ không thể khôi phục lại văn bản gốc.



Viện Tiêu chuẩn và Công nghệ Quốc gia (NIST) đã xuất bản Tiêu chuẩn Mã hóa Dữ liệu (DES) vào năm 1977. DES là một thuật toán mã hóa đối xứng sử dụng khóa có kích thước 56 bit. Năm 1997, một thử thách để phá vỡ một tin nhắn được mã hóa bằng DES đã được giải quyết. Do đó, đã được chứng minh rằng việc sử dụng tìm kiếm brute-force để tìm khóa và phá vỡ một tin nhắn được mã hóa bằng DES đã trở nên khả thi. Năm 1998, một khóa DES đã bị phá vỡ trong 56 giờ. Những trường hợp này cho thấy DES không còn được coi là an toàn.

NIST đã xuất bản Tiêu chuẩn Mã hóa Nâng cao (AES) vào năm 2001. Giống như DES, đó là một thuật toán mã hóa đối xứng; tuy nhiên, nó sử dụng khóa có kích thước 128, 192 hoặc 256 bit và vẫn được coi là an toàn và được sử dụng đến ngày nay. AES lặp lại bốn phép biến đổi sau đây nhiều lần:

SubBytes (state): Phép biến đổi này tìm kiếm từng byte trong một bảng thay thế (S-box) được cung cấp và thay thế nó bằng giá trị tương ứng. Trạng thái là 16 byte, tức là 128 bit, được lưu trữ trong một mảng 4x4. ShiftRows (state): Hàng thứ hai được dịch chuyển một chỗ, hàng thứ ba được dịch chuyển hai chỗ, và hàng thứ tư được dịch chuyển ba chỗ. Điều này được hiển thị trong hình dưới đây. MixColumns (state): Mỗi cột được nhân với một ma trận cố định (một mảng 4x4). AddRoundKey (state): Một khóa vòng được thêm vào trạng thái bằng phép XOR.



Tổng số vòng biến đổi phụ thuộc vào kích thước khóa.

Đừng lo lắng nếu bạn thấy điều này khó hiểu vì nó thực sự là như vậy! Mục đích của chúng tôi không phải là học chi tiết về cách AES hoạt động hoặc triển khai nó như một thư viện lập trình; mục đích là để đánh giá sự khác biệt về độ phức tạp giữa các thuật toán mã hóa cổ điển và hiện đại. Nếu bạn tò mò muốn tìm hiểu chi tiết, bạn có thể kiểm tra các thông số kỹ thuật AES, bao gồm mã giả và ví dụ trong tiêu chuẩn được xuất bản của nó, FIPS PUB 197.

Ngoài AES, nhiều thuật toán mã hóa đối xứng khác được coi là an toàn. Đây là danh sách các thuật toán mã hóa đối xứng được hỗ trợ bởi GPG (GnuPG) 2.37.7, ví dụ:

Thuật toán mã hóa Ghi chú

AES, AES192 và AES 256 AES với kích thước khóa 128, 192 và 256 bit

IDEA International Data Encryption Algorithm (IDEA) 3DES Triple DES (Data Encryption Standard) và dựa trên DES. Chúng ta nên lưu ý rằng 3DES sẽ bị loại bỏ vào năm 2023 và bị cấm vào năm 2024.

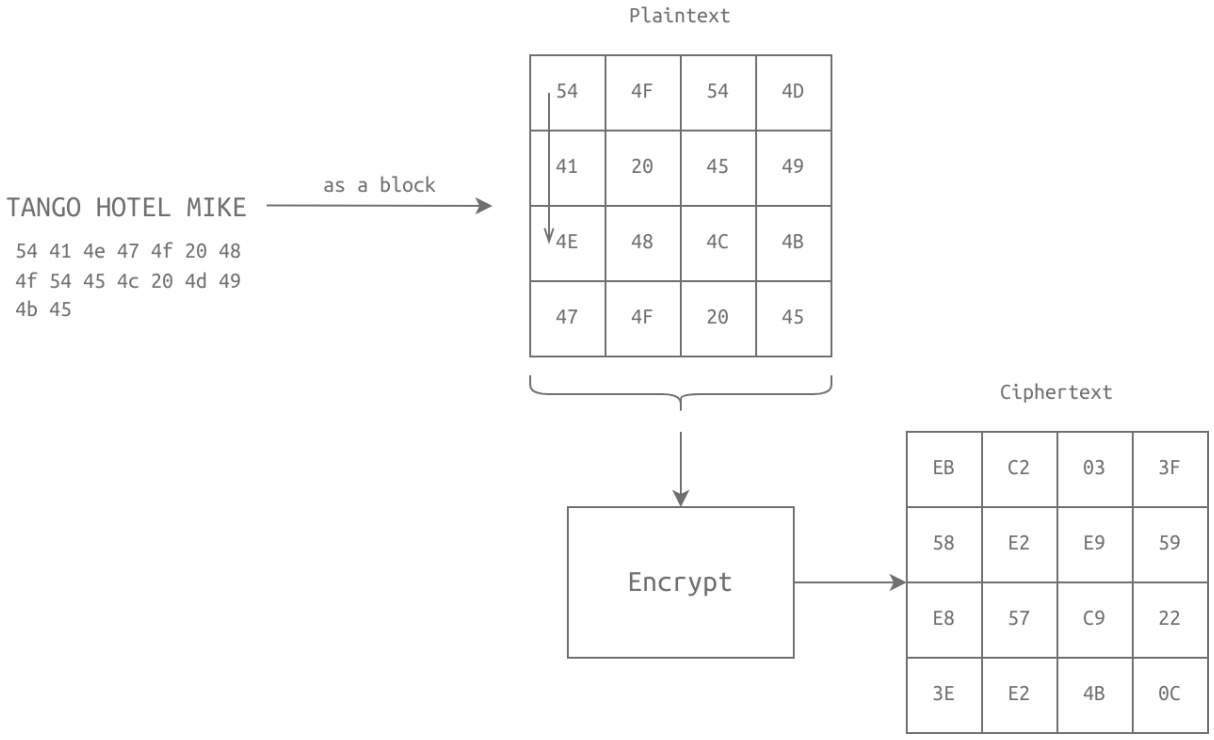
CAST5 Còn được gọi là CAST-128. Một số nguồn cho biết CASE là viết tắt của tên tác giả của nó: Carlisle Adams và Stafford Tavares.

BLOWFISH Được thiết kế bởi Bruce Schneier

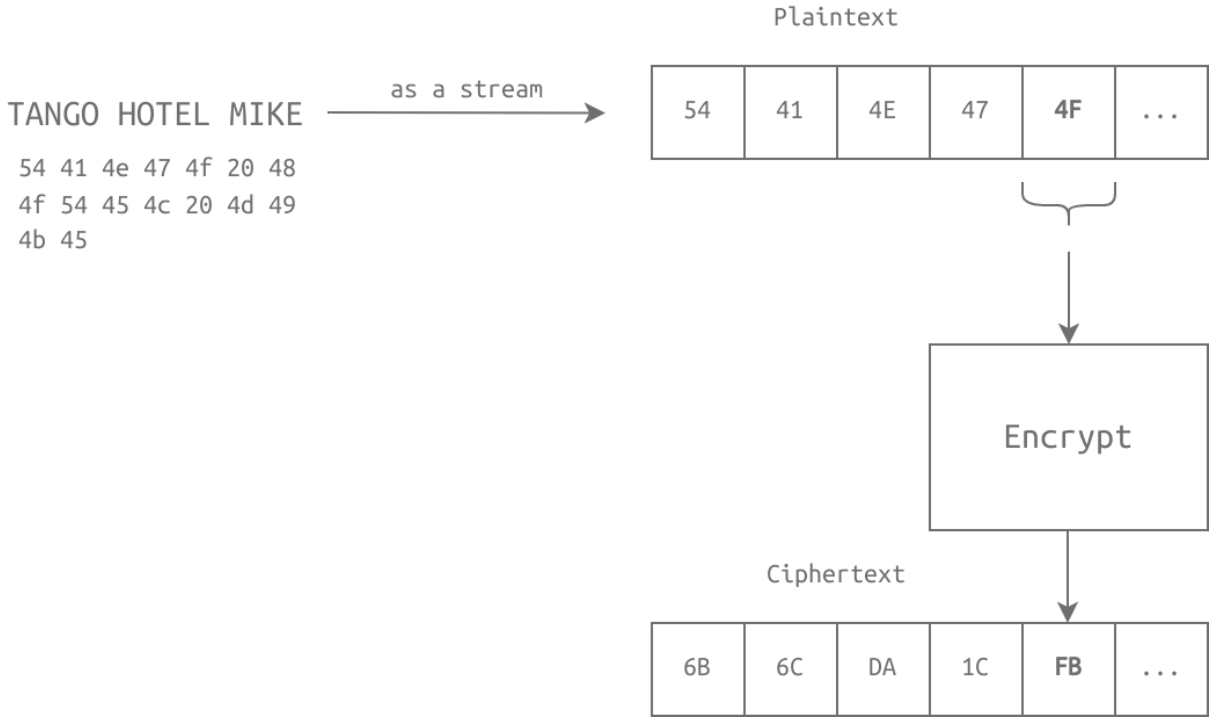
TWOFISH Được thiết kế bởi Bruce Schneier và được phát triển từ Blowfish CAMELLIA128, CAMELLIA192 và CAMELLIA256 Được thiết kế bởi Mitsubishi Electric và NTT tại Nhật Bản. Tên của nó được lấy từ hoa camellia japonica.

Tất cả các thuật toán được đề cập cho đến nay đều là thuật toán mã hóa đối xứng khối. Một thuật toán mã hóa khối chuyển đổi đầu vào (văn bản gốc) thành các khối và mã hóa từng khối. Một khối thường là 128 bit. Trong hình dưới đây, chúng ta muốn mã hóa văn bản gốc “TANGO HOTEL MIKE”, tổng cộng 16 ký tự. Bước đầu

tiên là biểu diễn nó dưới dạng nhị phân. Nếu chúng ta sử dụng ASCII, “T” là 0x54 ở định dạng thập lục phân, “A” là 0x41 và cứ như vậy. Mỗi hai chữ số thập lục phân tương ứng với 8 bit và đại diện cho một byte. Một khối 128 bit thực tế là 16 byte và được biểu diễn trong một mảng 4x4. Khối 128 bit được cung cấp như một đơn vị cho phương pháp mã hóa.



Loại thuật toán mã hóa đối xứng khác là thuật toán mã hóa dòng, mã hóa văn bản gốc byte theo byte. Hãy xem xét trường hợp chúng ta muốn mã hóa thông điệp “TANGO HOTEL MIKE”; mỗi ký tự cần được chuyển đổi sang biểu diễn nhị phân của nó. Nếu chúng ta sử dụng ASCII, “T” là 0x54 ở định dạng thập lục phân, trong khi “A” là 0x41 và cứ như vậy. Phương pháp mã hóa sẽ xử lý một byte một lần. Điều này được hiển thị trong hình dưới đây.



Thuật toán mã hóa đối xứng giải quyết nhiều vấn đề bảo mật được thảo luận trong phòng Nguyên tắc Bảo mật. Hãy giả sử rằng Alice và Bob gặp nhau và chọn một thuật toán mã hóa và đồng ý về một khóa cụ thể. Chúng ta giả định rằng thuật toán mã hóa được chọn là an toàn và khóa bí mật được giữ an toàn. Hãy xem xét những gì chúng ta có thể đạt được:

Tính bảo mật: Nếu Eve chặn được tin nhắn được mã hóa, cô ấy sẽ không thể khôi phục lại văn bản gốc. Do đó, tất cả các tin nhắn trao đổi giữa Alice và Bob đều được bảo mật miễn là chúng được gửi dưới dạng được mã hóa. Tính toàn vẹn: Khi Bob nhận được một tin nhắn được mã hóa và giải mã thành công bằng khóa mà anh ấy đã đồng ý với Alice, Bob có thể chắc chắn rằng không ai có thể can thiệp vào tin nhắn trên kênh. Khi sử dụng các thuật toán mã hóa hiện đại và an toàn, bất kỳ sửa đổi nhỏ nào trên văn bản được mã hóa đều sẽ ngăn chặn việc giải mã thành công hoặc dẫn đến văn bản gốc trở nên vô nghĩa. Tính xác thực: Việc giải mã văn bản được mã hóa bằng khóa bí mật cũng chứng minh tính xác thực của tin nhắn vì chỉ có Alice và Bob biết khóa bí mật. Chúng ta mới chỉ bắt đầu, và chúng ta biết cách duy trì tính bảo mật, kiểm tra tính toàn vẹn và đảm bảo tính xác thực của các tin nhắn trao đổi. Các phương pháp thực tế và hiệu quả hơn sẽ được trình bày trong các nhiệm vụ sau. Câu hỏi, trong lúc này, là liệu điều này có thể mở rộng.

Với Alice và Bob, chúng ta cần một khóa. Nếu chúng ta có Alice, Bob và Charlie, chúng ta cần ba khóa: một cho Alice và Bob, một cho Alice và Charlie và một cho Bob và Charlie. Tuy nhiên, số lượng khóa tăng nhanh; giao tiếp giữa 100 người dùng

yêu cầu gần 5000 khóa bí mật khác nhau. (Nếu bạn tò mò về toán học đằng sau đó, đó là 99 + 98 + 97 + … + 1 = 4950).

Hơn nữa, nếu một hệ thống bị xâm nhập, họ cần tạo ra các khóa mới để sử dụng với 99 người dùng khác. Vấn đề khác sẽ là tìm một kênh an toàn để trao đổi khóa với tất cả các người dùng khác. Rõ ràng, điều này nhanh chóng trở nên quá lớn.

Có nhiều chương trình có sẵn cho mã hóa đối xứng. Chúng ta sẽ tập trung vào hai chương trình, được sử dụng rộng rãi cho mã hóa không đối xứng:

GNU Privacy Guard Dự án OpenSSL GNU Privacy Guard GNU Privacy Guard, còn được gọi là GnuPG hoặc GPG, thực hiện tiêu chuẩn OpenPGP.

Chúng ta có thể mã hóa một tệp bằng GnuPG (GPG) bằng lệnh sau:

gpg --symmetric --cipher-algo CIPHER message.txt, trong đó CIPHER là tên của thuật toán mã hóa. Bạn có thể kiểm tra các thuật toán mã hóa được hỗ trợ bằng lệnh gpg --version. Tệp được mã hóa sẽ được lưu trữ dưới dạng message.txt.gpg.

Đầu ra mặc định là định dạng OpenPGP nhị phân; tuy nhiên, nếu bạn muốn tạo đầu ra ASCII armoured, có thể mở trong bất kỳ trình soạn thảo văn bản nào, bạn nên thêm tùy chọn --armor. Ví dụ: gpg --armor --symmetric --cipher-algo CIPHER message.txt.

Bạn có thể giải mã bằng lệnh sau:

gpg --output original_message.txt --decrypt message.gpg

Dự án OpenSSL Dự án OpenSSL duy trì phần mềm OpenSSL.

Chúng ta có thể mã hóa một tệp bằng OpenSSL bằng lệnh sau:

openssl aes-256-cbc -e -in message.txt -out encrypted_message

Chúng ta có thể giải mã tệp kết quả bằng lệnh sau:

openssl aes-256-cbc -d -in encrypted_message -out original_message.txt

Để làm cho mã hóa an toàn hơn và chống lại các cuộc tấn công brute-force, chúng ta có thể thêm -pbkdf2 để sử dụng Hàm phát sinh khóa dựa trên mật khẩu 2 (PBKDF2); hơn nữa, chúng ta có thể chỉ định số lần lặp lại trên mật khẩu để tạo ra khóa mã hóa bằng cách sử dụng -iter NUMBER. Để lặp lại 10.000 lần, lệnh trước đó sẽ trở thành:

openssl aes-256-cbc -pbkdf2 -iter 10000 -e -in message.txt -out encrypted_message

Do đó, lệnh giải mã trở thành:

openssl aes-256-cbc -pbkdf2 -iter 10000 -d -in encrypted_message -out original_message.txt

III. Asymmetric Encryption

Thuật toán mã hóa đối xứng yêu cầu người dùng tìm một kênh an toàn để trao đổi khóa. Bằng kênh an toàn, chúng ta chủ yếu quan tâm đến tính bảo mật và tính toàn vẹn. Nói cách khác, chúng ta cần một kênh mà không có bên thứ ba nào có thể nghe trộm và đọc lưu lượng; hơn nữa, không ai có thể thay đổi các tin nhắn và dữ liệu đã gửi.

Mã hóa không đối xứng cho phép trao đổi các tin nhắn được mã hóa mà không cần một kênh an toàn; chúng ta chỉ cần một kênh đáng tin cậy. Bằng kênh đáng tin cậy, chúng ta có nghĩa là chúng ta chủ yếu quan tâm đến tính toàn vẹn của kênh và không phải tính bảo mật.

Khi sử dụng một thuật toán mã hóa không đối xứng, chúng ta sẽ tạo ra một cặp khóa: một khóa công khai và một khóa bí mật. Khóa công khai được chia sẻ với thế giới, hoặc cụ thể hơn, với những người muốn trao đổi tin nhắn với chúng ta một cách an toàn. Khóa bí mật phải được lưu trữ an toàn, và chúng ta không bao giờ để bất kỳ ai truy cập vào nó. Hơn nữa, không khả thi để suy ra khóa bí mật mặc dù biết khóa công khai.

Cặp khóa này hoạt động như thế nào?

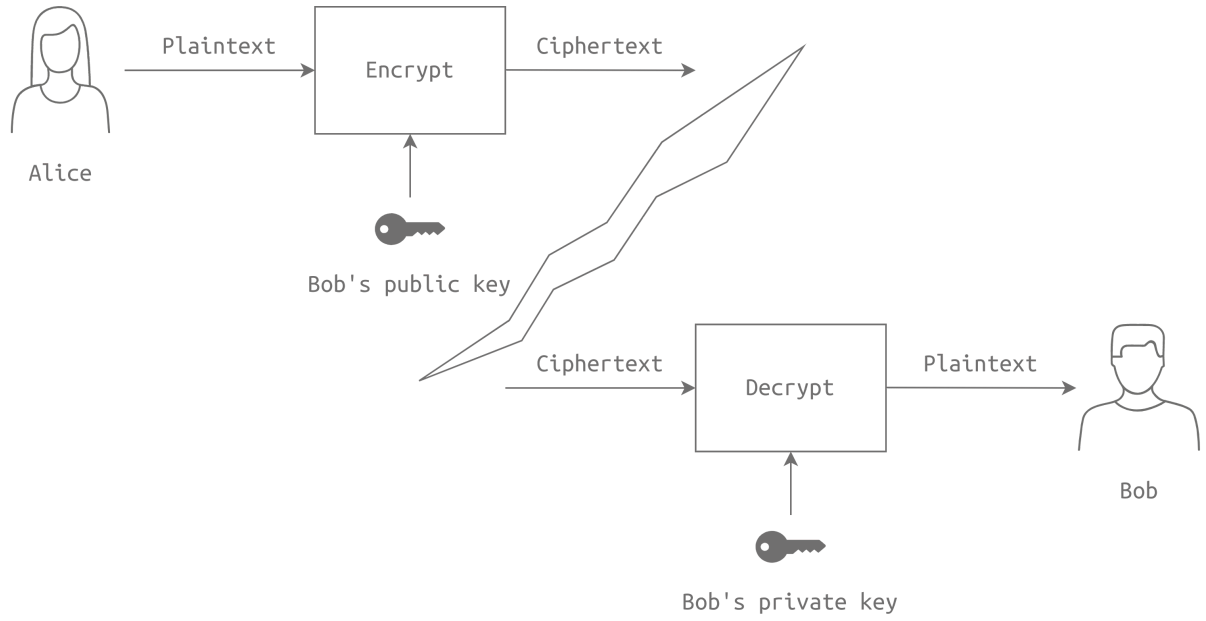
Nếu một tin nhắn được mã hóa bằng một khóa, nó có thể được giải mã bằng khóa khác. Nói cách khác:

Nếu Alice mã hóa một tin nhắn bằng khóa công khai của Bob, nó chỉ có thể được giải mã bằng khóa bí mật của Bob. Ngược lại, nếu Bob mã hóa một tin nhắn bằng khóa bí mật của mình, nó chỉ có thể được giải mã bằng khóa công khai của Bob.

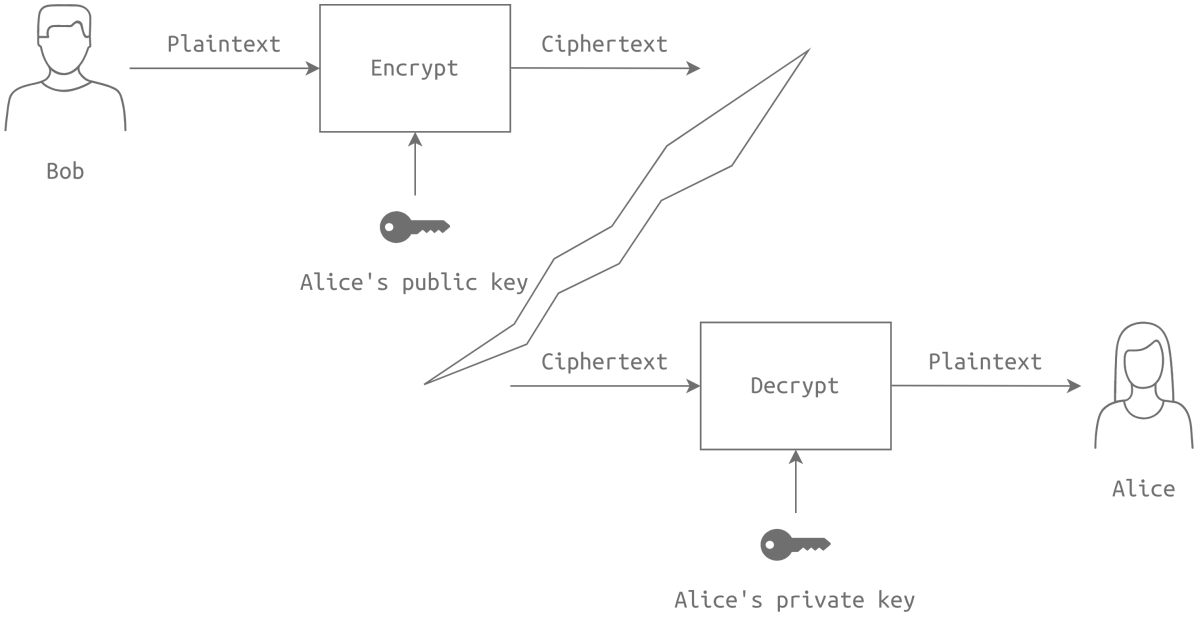
Tính bảo mật

Chúng ta có thể sử dụng mã hóa không đối xứng để đạt được tính bảo mật bằng cách mã hóa các tin nhắn bằng khóa công khai của người nhận. Trong hai hình dưới đây, chúng ta có thể thấy rằng:

Alice muốn đảm bảo tính bảo mật trong việc giao tiếp với Bob. Cô ấy mã hóa tin nhắn bằng khóa công khai của Bob và Bob giải mã chúng bằng khóa bí mật của anh ấy. Khóa công khai của Bob được mong đợi sẽ được xuất bản trên cơ sở dữ liệu công khai hoặc trên trang web của anh ấy, ví dụ như.



Khi Bob muốn trả lời cho Alice, anh ấy mã hóa tin nhắn của mình bằng khóa công khai của Alice, và Alice có thể giải mã chúng bằng khóa bí mật của mình.



Nói cách khác, trở nên dễ dàng để giao tiếp với Alice và Bob trong khi đảm bảo tính bảo mật của các tin nhắn. Yêu cầu duy nhất là tất cả các bên đều có khóa công khai của họ có sẵn cho người gửi quan tâm.

Lưu ý: Trong thực tế, các thuật toán mã hóa đối xứng cho phép các hoạt động nhanh hơn so với mã hóa không đối xứng; do đó, chúng ta sẽ bàn về cách sử dụng tốt nhất của cả hai thế giới sau.

Tính toàn vẹn, tính xác thực và không thể chối bỏ

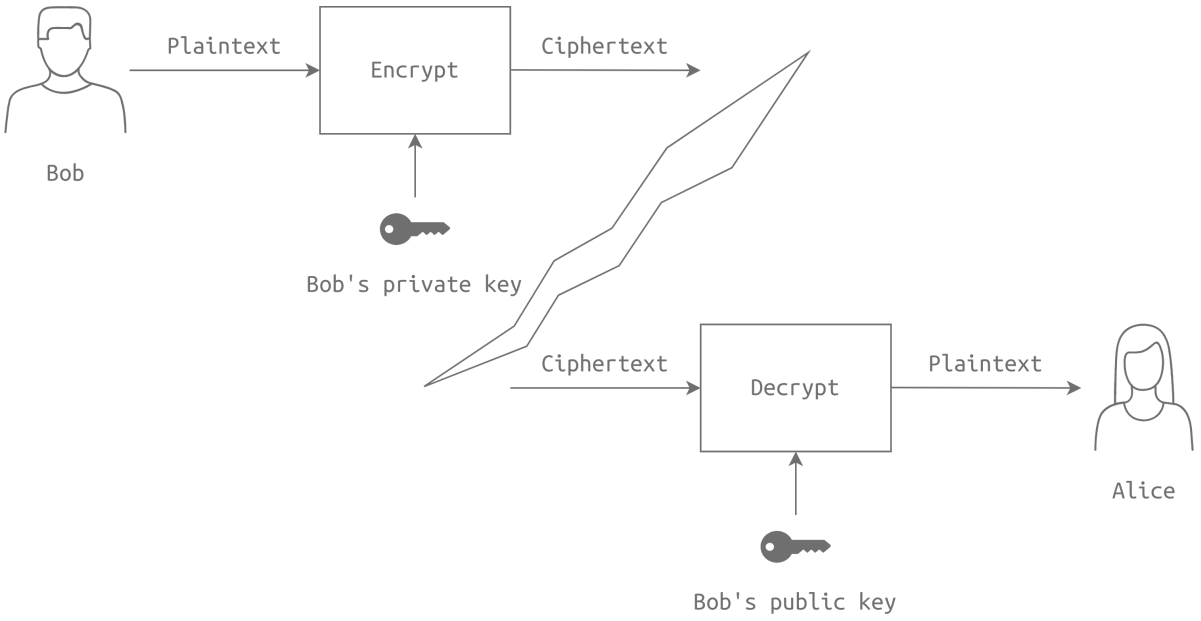
Ngoài tính bảo mật, mã hóa không đối xứng có thể giải quyết tính toàn vẹn, tính xác thực và không thể chối bỏ. Hãy giả sử rằng Bob muốn đưa ra một tuyên bố và muốn cho tất cả mọi người xác nhận rằng tuyên bố này thực sự đến từ anh ấy. Bob cần mã hóa tin nhắn bằng khóa bí mật của mình; người nhận có thể giải mã nó bằng khóa công khai của Bob. Nếu tin nhắn được giải mã thành công bằng khóa công khai của Bob, điều đó có nghĩa là tin nhắn đã được mã hóa bằng khóa bí mật của Bob. (Trong thực tế, anh ấy sẽ mã hóa một băm của tin nhắn gốc. Chúng tôi sẽ giải thích thêm về điều này sau.)

Việc giải mã thành công bằng khóa công khai của Bob dẫn đến một vài kết luận thú vị.

Thứ nhất, tin nhắn không bị thay đổi trên đường đi (kênh truyền thông); điều này chứng minh tính toàn vẹn của tin nhắn.

Thứ hai, biết rằng không ai có quyền truy cập vào khóa bí mật của Bob, chúng ta có thể chắc chắn rằng tin nhắn này thực sự đến từ Bob; điều này chứng minh tính xác thực của tin nhắn.

Cuối cùng, vì không ai ngoại trừ Bob có quyền truy cập vào khóa bí mật của Bob, Bob không thể phủ nhận việc gửi tin nhắn này; điều này thiết lập tính không thể chối bỏ.



Chúng ta đã thấy làm thế nào mã hóa không đối xứng có thể giúp thiết lập tính bảo mật, tính toàn vẹn, tính xác thực và không thể chối bỏ. Trong các kịch bản thực tế, mã hóa không đối xứng có thể khá chậm để mã hóa các tệp lớn và lượng dữ liệu lớn. Trong một nhiệm vụ khác, chúng ta sẽ thấy làm thế nào chúng ta có thể sử dụng mã hóa không đối xứng kết hợp với mã hóa đối xứng để đạt được các mục tiêu bảo mật này một cách tương đối nhanh chóng.

RSA RSA được đặt theo tên các nhà phát minh của nó, Rivest, Shamir và Adleman. Nó hoạt động như sau:

Chọn hai số nguyên tố ngẫu nhiên, p và q. Tính N = p × q.

Chọn hai số nguyên e và d sao cho e × d = 1 mod φ(N), trong đó φ(N) = N − p − q + 1. Bước này sẽ cho phép chúng ta tạo khóa công khai (N,e) và khóa bí mật (N,d).

Người gửi có thể mã hóa giá trị x bằng cách tính y = x^e mod N. (Modulus) Người nhận có thể giải mã y bằng cách tính x = y^d mod N. Lưu ý rằng y^d = x^(ed) = x^(kφ(N) + 1) = (x^(φ(N))^k × x = x. Bước này giải thích tại sao chúng ta đặt ràng buộc trên sự lựa chọn của e và d.

Đừng lo lắng nếu các phương trình toán học trên trông quá phức tạp; bạn không cần toán học để có thể sử dụng RSA, vì nó có sẵn thông qua các chương trình và thư viện lập trình.

Bảo mật RSA phụ thuộc vào việc phân tích thành thừa số là một vấn đề khó. Việc nhân p với q rất dễ; tuy nhiên, việc tìm p và q cho trước N mất nhiều thời gian. Hơn nữa, để đảm bảo tính bảo mật, p và q nên là các số khá lớn, ví dụ, mỗi số có 1024 bit (đó là một số có hơn 300 chữ số). Quan trọng là lưu ý rằng RSA phụ thuộc vào

việc tạo số ngẫu nhiên an toàn, giống như các thuật toán mã hóa không đối xứng khác. Nếu một kẻ thù có thể đoán được p và q, toàn bộ hệ thống sẽ bị coi là không an toàn.

Hãy xem xét ví dụ thực tế sau đây.

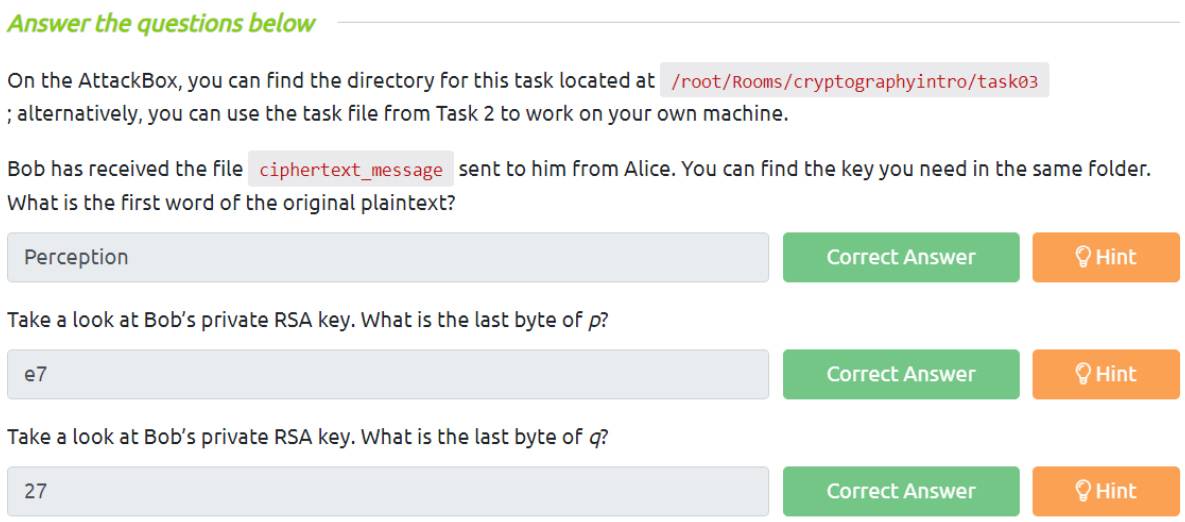
Bob chọn hai số nguyên tố: p = 157 và q = 199. Anh ấy tính N = 31243. Với φ(N) = N − p − q + 1 = 31243 − 157 − 199 + 1 = 30888, Bob chọn e = 163 và d = 379 trong đó e × d = 163 × 379 = 61777 và 61777 mod 30888 = 1. Khóa công khai là (31243,163) và khóa bí mật là (31243,379). Giả sử giá trị cần mã hóa là x = 13, sau đó Alice sẽ tính toán và gửi y = xe mod N = 13163 mod 31243 = 16342. Bob sẽ giải mã giá trị nhận được bằng cách tính x = yd mod N = 16341379 mod 31243 = 13.

Ví dụ trước đó là để hiểu rõ hơn về toán học đằng sau nó. Để xem các giá trị thực cho p và q, hãy tạo một cặp khóa thực sự bằng cách sử dụng một công cụ như openssl.

Chúng ta đã thực hiện ba lệnh:

openssl genrsa -out private-key.pem 2048: Sử dụng openssl, chúng ta đã sử dụng genrsa để tạo khóa bí mật RSA. Sử dụng -out, chúng ta chỉ định rằng khóa bí mật kết quả được lưu trữ dưới dạng private-key.pem. Chúng tôi đã thêm 2048 để chỉ định kích thước khóa là 2048 bit. openssl rsa -in private-key.pem -pubout -out public-key.pem: Sử dụng openssl, chúng ta chỉ định rằng chúng ta đang sử dụng thuật toán RSA với tùy chọn rsa. Chúng tôi chỉ định rằng chúng tôi muốn lấy khóa công khai bằng cách sử dụng -pubout. Cuối cùng, chúng tôi đặt khóa bí mật làm đầu vào bằng cách sử dụng -in private-key.pem và lưu đầu ra bằng cách sử dụng -out public-key.pem. openssl rsa -in private-key.pem -text -noout: Chúng tôi tò mò muốn xem các biến RSA thực sự, vì vậy chúng tôi đã sử dụng -text -noout. Các giá trị của p, q, N, e và d lần lượt là prime1, prime2, modulus, publicExponent và privateExponent. Nếu chúng ta đã có khóa công khai của người nhận, chúng ta có thể mã hóa nó bằng lệnh openssl pkeyutl -encrypt -in plaintext.txt -out ciphertext -inkey public-key.pem -pubin

Người nhận có thể giải mã nó bằng lệnh openssl pkeyutl -decrypt -in ciphertext -inkey private-key.pem -out decrypted.txt



IV. Diffie-Hellman Key Exchange

Alice và Bob có thể trao đổi thông tin qua một kênh không an toàn. Kênh không an toàn có nghĩa là có những kẻ nghe trộm có thể đọc được các tin nhắn được trao đổi trên kênh này. Làm thế nào Alice và Bob có thể đồng ý về một khóa bí mật trong một cài đặt như vậy? Một cách là sử dụng trao đổi khóa Diffie-Hellman.

Diffie-Hellman là một thuật toán mã hóa không đối xứng. Nó cho phép trao đổi một khóa bí mật qua một kênh công khai. Chúng ta sẽ bỏ qua nền tảng toán học modular và cung cấp một ví dụ số đơn giản. Chúng ta cần hai phép toán toán học: lũy thừa và phần dư. xp, tức là x mũ p, là x nhân với chính nó p lần. Hơn nữa, x mod m, tức là phần dư của x khi chia cho m, là phần dư của phép chia x cho m.

Alice và Bob đồng ý về q và g. Để làm việc này, q phải là một số nguyên tố, và g là một số nhỏ hơn q thỏa mãn một số điều kiện nhất định. (Trong toán học modular, g là một bộ phát sinh.) Trong ví dụ này, chúng ta lấy q = 29 và g = 3. Alice chọn một số ngẫu nhiên a nhỏ hơn q. Cô ấy tính A = (ga) mod q. Số a phải được giữ bí mật; tuy nhiên, A được gửi đến Bob. Hãy giả sử rằng Alice chọn số a = 13 và tính toán A = 3^13%29 = 19 và gửi nó đến Bob.

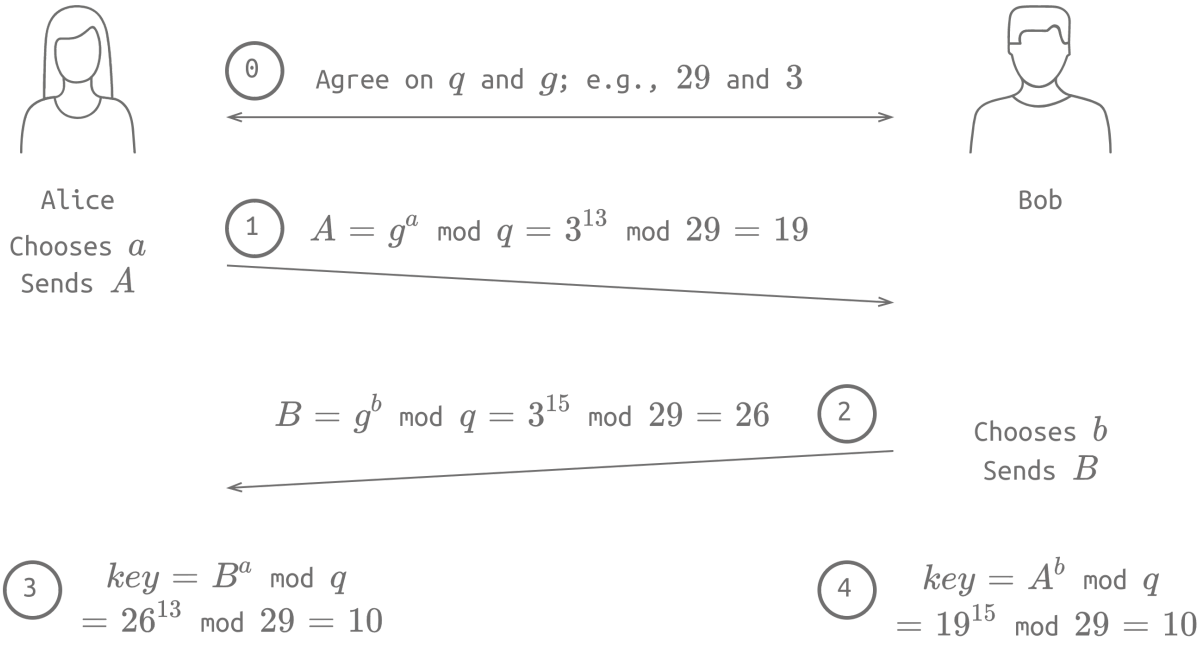
Bob chọn một số ngẫu nhiên b nhỏ hơn q. Anh ấy tính B = (gb) mod q. Bob phải giữ b bí mật; tuy nhiên, anh ấy gửi B đến Alice. Hãy xem xét trường hợp Bob chọn số b = 15 và tính toán B = 3^15%29 = 26. Anh ấy tiếp tục gửi nó đến Alice.

Alice nhận được B và tính toán key = Ba mod q. Ví dụ số key = 2613 mod 29 = 10.

Bob nhận được A và tính toán key = Ab mod q. Ví dụ số key = 1915 mod 29 = 10.

Chúng ta có thể thấy rằng Alice và Bob đã đạt được cùng một khóa.

Mặc dù một kẻ nghe trộm đã biết các giá trị của q, g, A và B, họ sẽ không thể tính toán được khóa bí mật mà Alice và Bob đã trao đổi. Các bước trên được tóm tắt trong hình dưới đây.

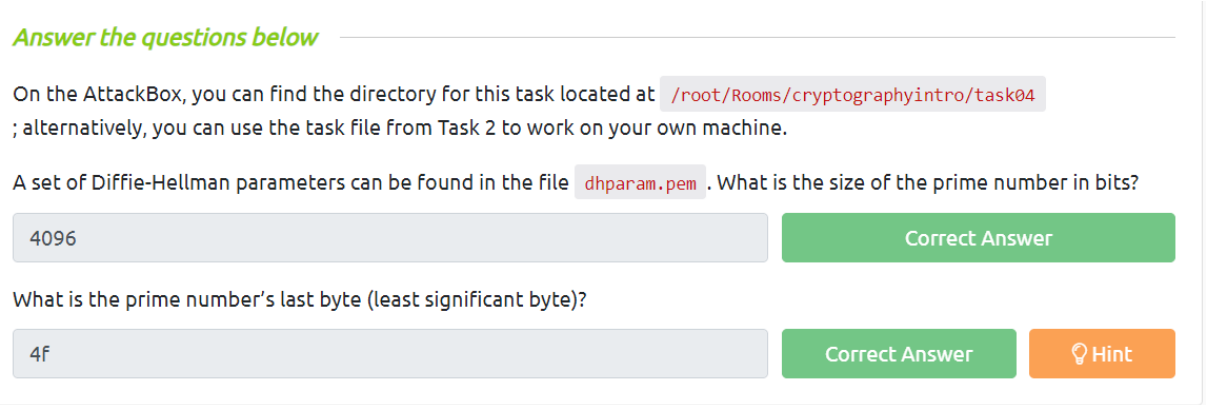


Những số mà chúng ta đã chọn làm cho việc tìm a và b dễ dàng, ngay cả khi không sử dụng máy tính, nhưng các ví dụ trong thực tế sẽ chọn một q có độ dài 256 bit. Trong hệ thập phân, đó là 115 với 75 số 0 bên phải (tôi cũng không biết cách đọc nó, nhưng tôi được cho biết nó được đọc là 115 quattuorvigintillion). Một q lớn như vậy sẽ làm cho việc tìm a hoặc b trở nên không khả thi mặc dù biết q, g, A và B.

Hãy xem xét các tham số Diffie-Hellman thực tế. Chúng ta có thể sử dụng openssl để tạo chúng; chúng ta cần chỉ định tùy chọn dhparam để cho biết chúng ta muốn tạo các tham số Diffie-Hellman cùng với kích thước được chỉ định theo bit, chẳng hạn như 2048 hoặc 4096.

Trong đầu ra console bên dưới, chúng ta có thể xem số nguyên tố P và bộ phát G bằng lệnh openssl dhparam -in dhparams.pem -text -noout. (Điều này tương tự như những gì chúng ta đã làm với khóa bí mật RSA.)

Thuật toán trao đổi khóa Diffie-Hellman cho phép hai bên đồng ý về một khóa bí mật qua một kênh không an toàn. Tuy nhiên, trao đổi khóa đã thảo luận dễ bị tấn công Man-in-the-Middle (MitM); một kẻ tấn công có thể trả lời Alice giả vờ là Bob và trả lời Bob giả vờ là Alice.



V. Hashing

Một hàm băm mật mã là một thuật toán lấy dữ liệu có kích thước tùy ý làm đầu vào và trả về một giá trị cố định, được gọi là bản tóm tắt tin nhắn hoặc checksum, làm đầu ra của nó. Ví dụ, sha256sum tính toán bản tóm tắt tin nhắn SHA256 (Secure Hash Algorithm 256). SHA256, như tên gọi, trả về một checksum có kích thước 256 bit (32 byte). Checksum này thường được viết bằng các chữ số thập lục phân. Biết rằng một chữ số thập lục phân đại diện cho 4 bit, checksum 256 bit có thể được biểu diễn dưới dạng 64 chữ số thập lục phân.

Trong đầu ra terminal bên dưới, chúng ta tính toán các giá trị băm SHA256 cho ba tệp có kích thước khác nhau: 4 byte, 275 MB và 5,2 GB. Sử dụng sha256sum để tính toán bản tóm tắt tin nhắn cho mỗi trong ba tệp, chúng ta nhận được ba giá trị hoàn toàn khác nhau và có vẻ ngẫu nhiên. Đáng chú ý rằng độ dài của bản tóm tắt tin nhắn hoặc checksum kết quả là giống nhau, bất kể tệp có kích thước nhỏ hay lớn đến đâu. Đặc biệt, tệp abc.txt có 4 byte và tệp 5,2 GB đều cho kết quả bản tóm tắt tin nhắn có độ dài bằng nhau độc lập với kích thước tệp.

Nhưng tại sao chúng ta cần một hàm như vậy? Có nhiều ứng dụng, đặc biệt là:

Lưu trữ mật khẩu: Thay vì lưu trữ mật khẩu dưới dạng văn bản thuần, một bản tóm tắt của mật khẩu được lưu trữ. Do đó, nếu xảy ra việc vi phạm dữ liệu, kẻ tấn công sẽ nhận được một danh sách các bản tóm tắt mật khẩu thay vì các mật khẩu gốc. (Trong thực tế, mật khẩu cũng được “muối”, như đã thảo luận trong một nhiệm vụ sau đó.)

Phát hiện sự thay đổi: Bất kỳ sự thay đổi nhỏ nào đối với tệp gốc đều dẫn đến một sự thay đổi đáng kể trong giá trị băm, tức là checksum.

Trong đầu ra terminal sau, chúng ta có hai tệp text1.txt và text2.txt, gần như giống nhau ngoại trừ (đúng nghĩa đen) một bit khác nhau; các chữ cái T và t khác nhau một bit trong biểu diễn ASCII của chúng. Mặc dù chúng ta chỉ đảo ngược một bit duy nhất, rõ ràng là các giá trị băm SHA256 hoàn toàn khác nhau. Do đó, nếu chúng ta

sử dụng một thuật toán hàm băm an toàn, chúng ta có thể dễ dàng xác nhận liệu có bất kỳ sự thay đổi nào đã xảy ra hay không. Điều này có thể giúp bảo vệ chống lại cả sự can thiệp có chủ ý và lỗi truyền tải tệp.

Một số thuật toán băm đang được sử dụng và vẫn được coi là an toàn bao gồm:

SHA224, SHA256, SHA384, SHA512

RIPEMD160

Một số hàm băm cũ hơn, chẳng hạn như MD5 (Message Digest 5) và SHA-1, đã bị phá vỡ mật mã. Bằng cách phá vỡ, chúng tôi có nghĩa là có thể tạo ra một tệp khác với cùng một checksum như một tệp đã cho. Điều này có nghĩa là chúng ta có thể tạo ra một va chạm băm. Nói cách khác, một kẻ tấn công có thể tạo ra một tin nhắn mới với một checksum đã cho và không thể phát hiện được sự can thiệp vào tệp hoặc tin nhắn.

HMAC

Hash-based message authentication code (HMAC) là một mã xác thực tin nhắn (MAC) sử dụng một khóa mật mã học ngoài hàm băm.

Theo RFC2104, HMAC cần:

khóa bí mật

bộ đệm trong (ipad) một chuỗi hằng số. (RFC2104 sử dụng byte 0x36 lặp lại B lần. Giá trị của B phụ thuộc vào hàm băm được chọn.)

bộ đệm ngoài (opad) một chuỗi hằng số. (RFC2104 sử dụng byte 0x5C lặp lại B lần.)

Việc tính toán HMAC tuân theo các bước sau như được hiển thị trong hình:

Thêm các số không vào khóa để làm cho nó có độ dài B, tức là làm cho độ dài của nó phù hợp với ipad.

Sử dụng XOR theo bit, được biểu thị bằng ⊕, tính toán key ⊕ ipad.

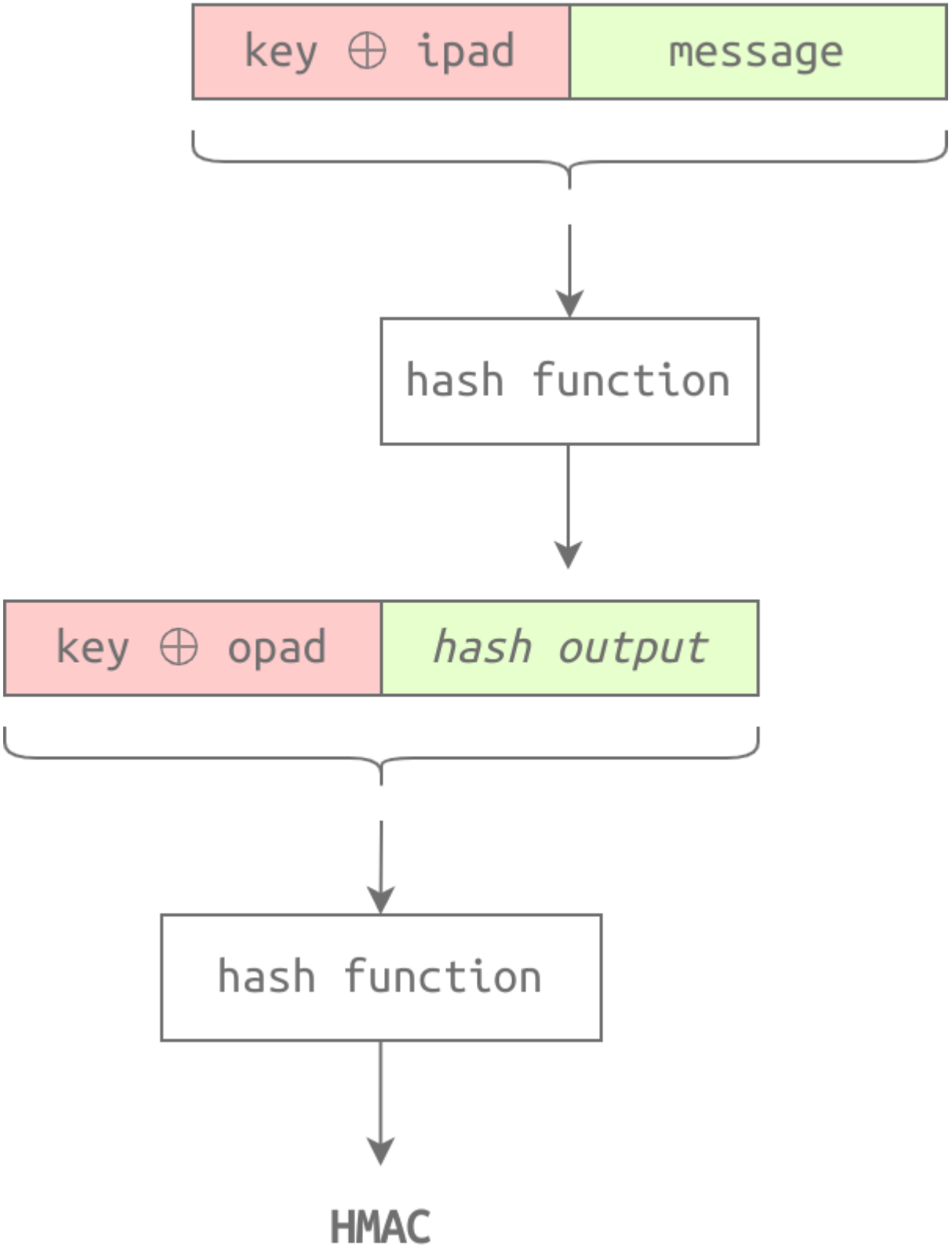
Thêm tin nhắn vào đầu ra XOR từ bước 2.

Áp dụng hàm băm cho luồng byte kết quả (trong bước 3).

Sử dụng XOR, tính toán key ⊕ opad.

Thêm đầu ra hàm băm từ bước 4 vào đầu ra XOR từ bước 5.

Áp dụng hàm băm cho luồng byte kết quả (trong bước 6) để có được HMAC.



Hình trên đại diện cho các bước được biểu thị trong công thức sau: H(K⊕opad,H(K⊕ipad,text)).

Để tính toán HMAC trên hệ thống Linux, bạn có thể sử dụng bất kỳ công cụ nào có sẵn như hmac256 (hoặc sha224hmac, sha256hmac, sha384hmac và sha512hmac, trong đó khóa bí mật được thêm sau tùy chọn --key).

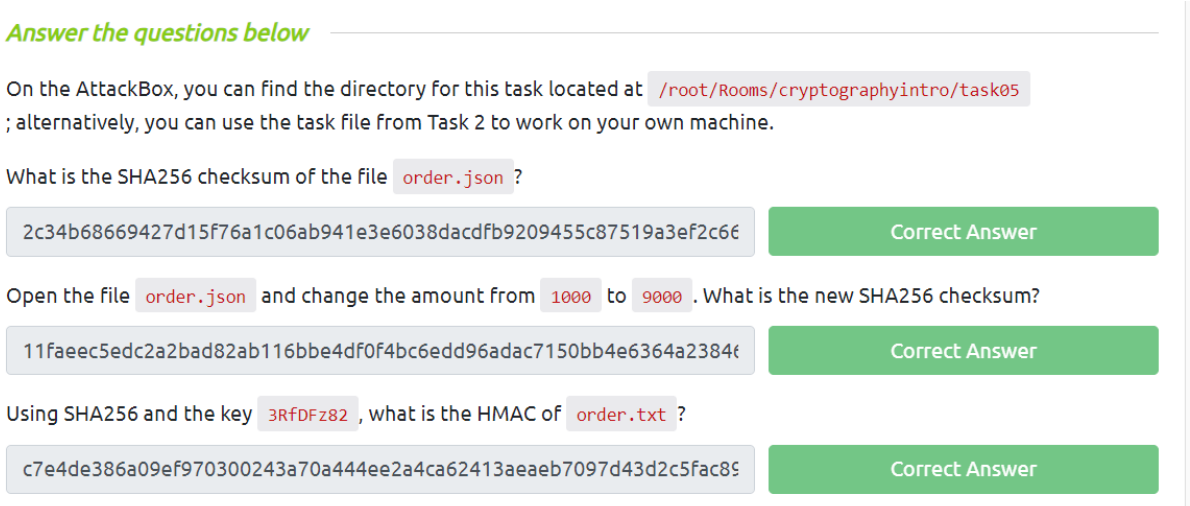
Dưới đây là một ví dụ về tính toán HMAC bằng cách sử dụng hmac256 và sha256hmac với hai khóa khác nhau.

user@TryHackMe**$**hmac256 s!Kr37 message.txt
3ec65b7e80c5bf2e623e52e0528f1c6a74f605b10616621ba1c22a89fb244e65 message.txt

user@TryHackMe
$hmac256 1234 message.txt
4b6a2783631180fca6128592e3d17fb5bff6b0e563ad8f1c6afc1050869e440f message.txt

user@TryHackMe
$sha256hmac message.txt --key s!Kr37
3ec65b7e80c5bf2e623e52e0528f1c6a74f605b10616621ba1c22a89fb244e65 message.txt

user@TryHackMe
$sha256hmac message.txt --key 1234
4b6a2783631180fca6128592e3d17fb5bff6b0e563ad8f1c6afc1050869e440f message.txt



VI. PKI and SSL/TLS

Sử dụng trao đổi khóa như trao đổi khóa Diffie-Hellman cho phép chúng ta đồng ý về một khóa bí mật dưới sự quan sát của kẻ nghe trộm. Khóa này có thể được sử dụng với một thuật toán mã hóa đối xứng để đảm bảo giao tiếp bảo mật. Tuy nhiên, trao đổi khóa mà chúng tôi đã mô tả trước đó không miễn nhiễm với cuộc tấn công Man-in-the-Middle (MITM). Lý do là Alice không có cách nào để đảm bảo rằng cô đang giao tiếp với Bob, và Bob cũng không có cách nào để đảm bảo rằng anh ấy đang giao tiếp với Alice khi trao đổi khóa bí mật.

Hãy xem xét hình dưới đây. Đây là một cuộc tấn công vào trao đổi khóa được giải thích trong nhiệm vụ Trao đổi khóa Diffie-Hellman. Các bước như sau:

Alice và Bob đồng ý về q và g. Bất kỳ ai lắng nghe trên kênh truyền thông đều có thể đọc hai giá trị này, bao gồm kẻ tấn công Mallory.

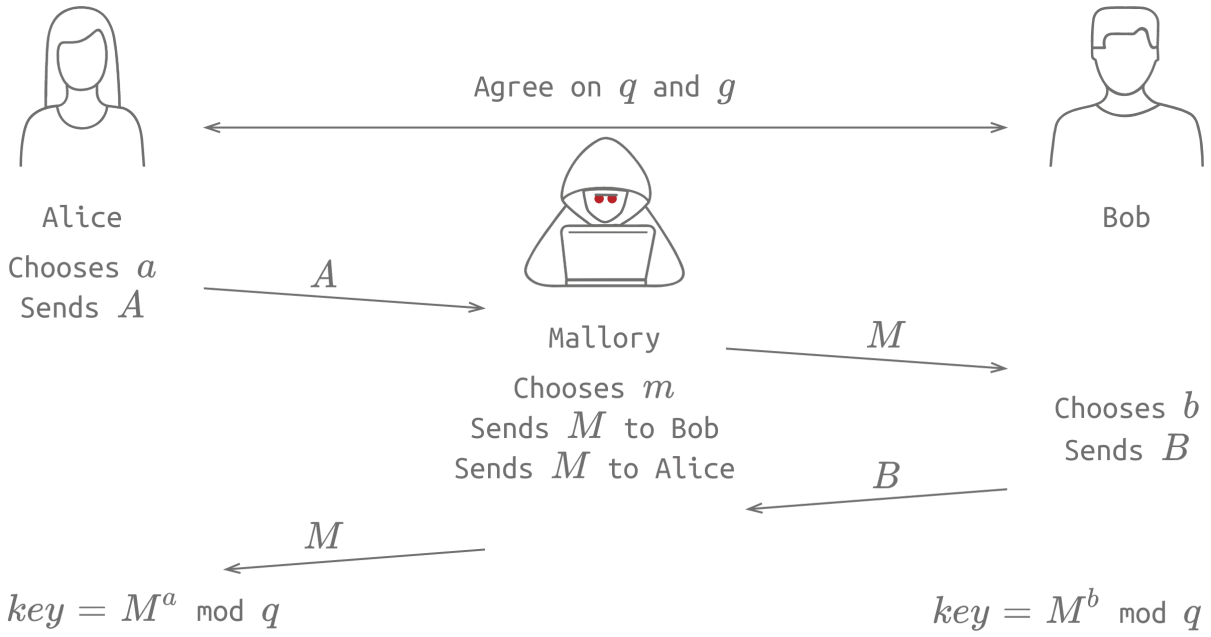
Như cô ấy thường làm, Alice chọn một biến ngẫu nhiên a, tính A (A = (ga) mod q) và gửi A đến Bob. Mallory đã đợi bước này và cô ấy đã chọn một biến ngẫu nhiên m và tính toán M tương ứng. Ngay khi Mallory nhận được A, cô ấy gửi M đến Bob, giả vờ như cô ấy là Alice.

Bob nhận M nghĩ rằng Alice đã gửi nó. Bob đã chọn một biến ngẫu nhiên b và tính B tương ứng; anh ấy gửi B đến Alice. Tương tự, Mallory chặn tin nhắn, đọc B và gửi M đến Alice thay vì.

Alice nhận được M và tính toán key = Ma mod q.

Bob nhận được M và tính toán key = Mb mod q.

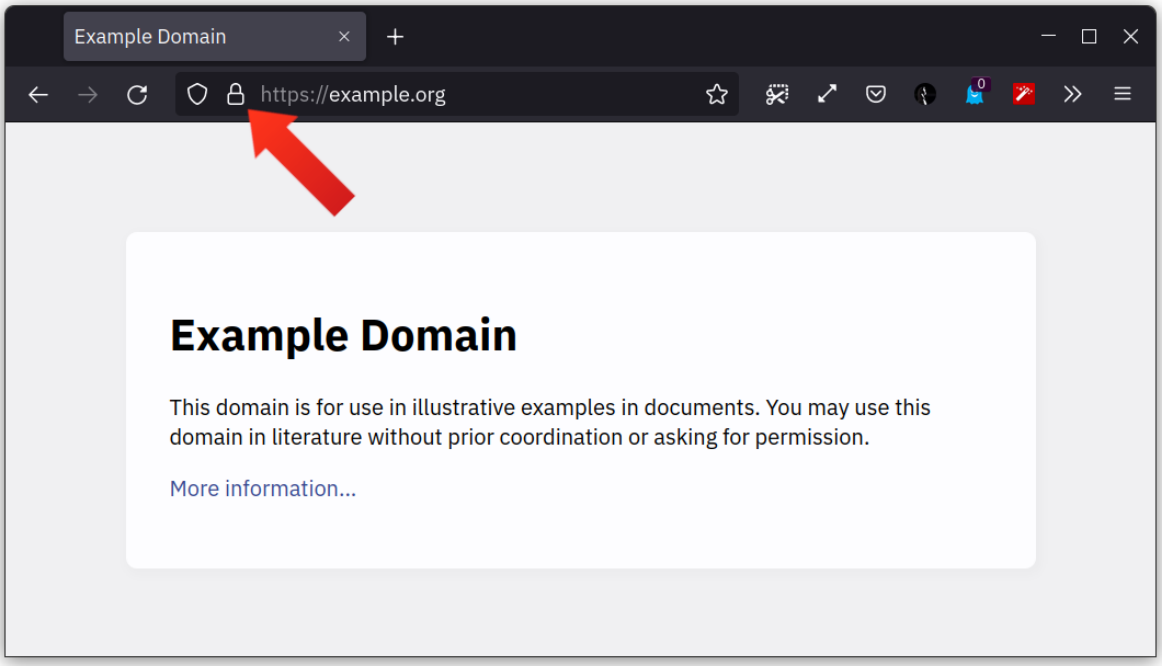
Alice và Bob tiếp tục giao tiếp, nghĩ rằng họ đang giao tiếp trực tiếp, không hề hay biết rằng họ đang giao tiếp với Mallory, người có thể đọc và sửa đổi các tin nhắn trước khi gửi chúng đến người nhận dự định.



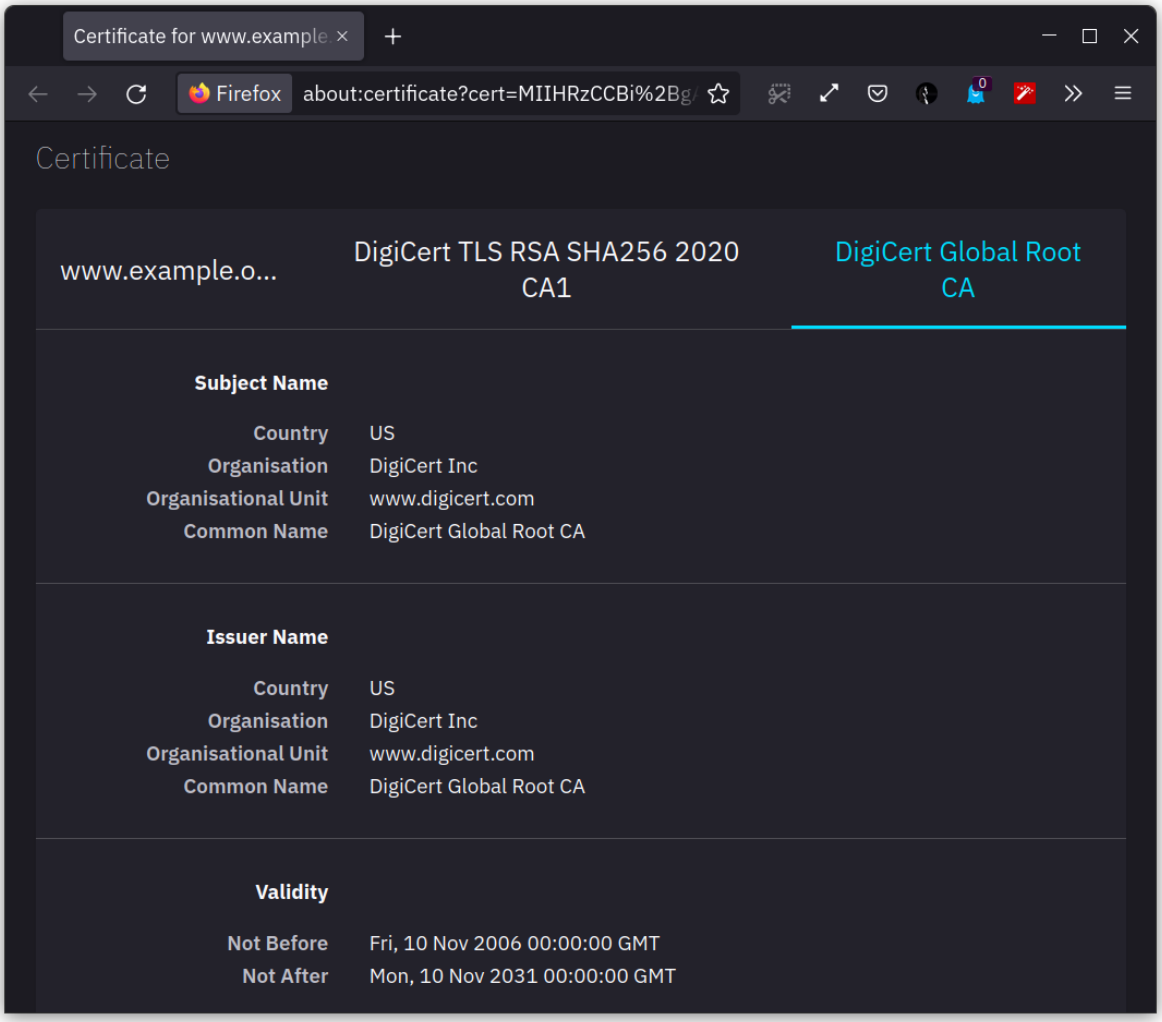
Khả năng bị tấn công này đòi hỏi cơ chế xác thực danh tính của bên kia. Điều này đưa chúng ta đến Public Key Infrastructure (PKI).

Hãy xem xét trường hợp bạn đang duyệt trang web example.org qua HTTPS. Làm thế nào để bạn có thể tự tin rằng bạn đang giao tiếp với máy chủ example.org? Nói cách khác, làm thế nào để bạn chắc chắn rằng không có kẻ thứ ba nào can thiệp vào các gói tin và thay đổi chúng trước khi chúng đến với bạn? Câu trả lời nằm trong chứng chỉ của trang web.

Hình dưới đây cho thấy trang mà chúng ta nhận được khi duyệt example.org. Hầu hết các trình duyệt đều biểu thị kết nối được mã hóa bằng một biểu tượng khóa. Biểu tượng khóa này cho thấy rằng kết nối được bảo mật qua HTTPS với một chứng chỉ hợp lệ.



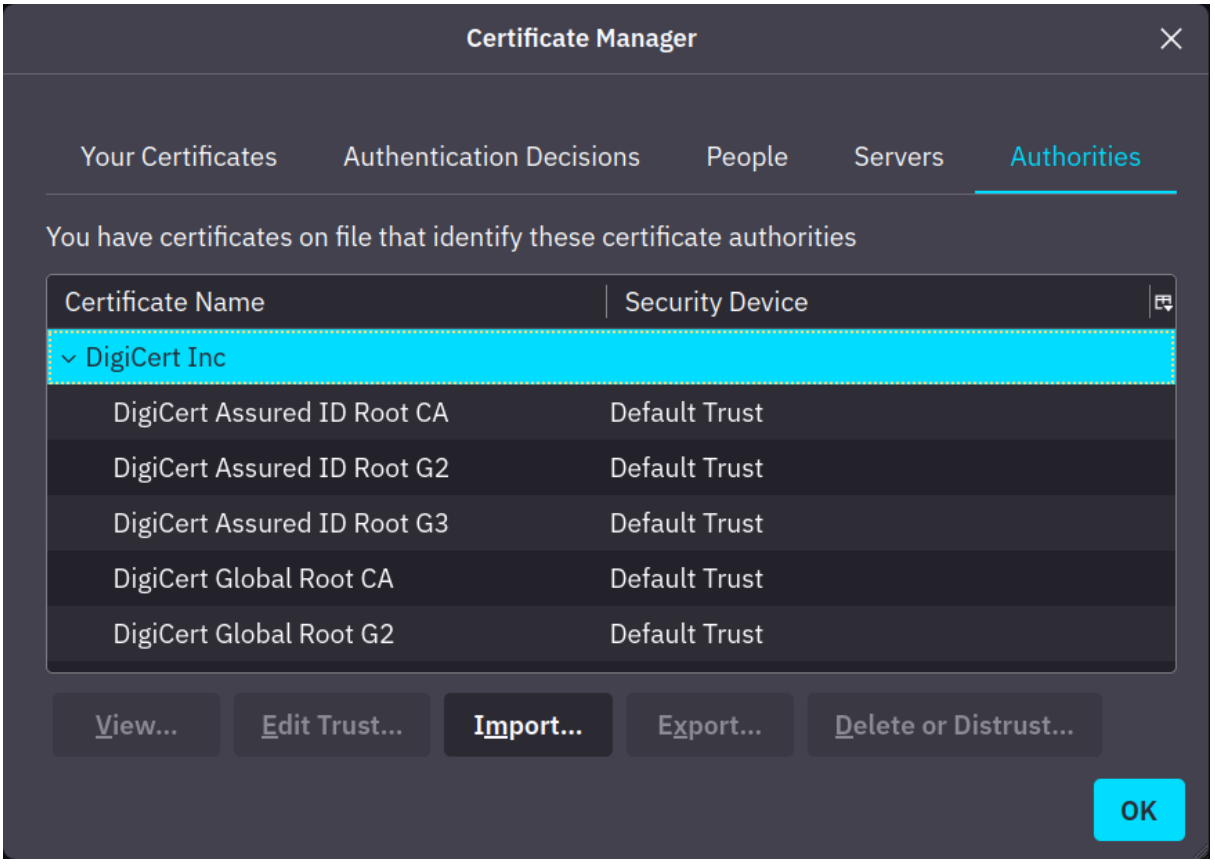
“Vào thời điểm viết bài, example.org sử dụng chứng chỉ được ký bởi DigiCert Inc., như được hiển thị trong hình dưới đây. Nói cách khác, DigiCert xác nhận rằng chứng chỉ này là hợp lệ (cho đến một ngày nhất định)



Để một chứng chỉ được ký bởi một Certificate Authority (CA), chúng ta cần thực hiện các bước sau:

1. Tạo Certificate Signing Request (CSR): Bạn tạo một chứng chỉ và gửi khóa công khai của bạn để được ký bởi một bên thứ ba.
2. Gửi CSR của bạn đến một Certificate Authority (CA): Mục đích là để CA ký chứng chỉ của bạn. Giải pháp thay thế và thường không an toàn sẽ là tự ký chứng chỉ của bạn.

Để quá trình này hoạt động, người nhận phải nhận ra và tin tưởng CA đã ký chứng chỉ. Và như chúng ta mong đợi, trình duyệt của chúng ta tin tưởng DigiCert Inc là một cơ quan ký chứng chỉ; nếu không, nó sẽ phát ra cảnh báo bảo mật thay vì tiếp tục truy cập trang web được yêu cầu.



Bạn có thể sử dụng openssl để tạo yêu cầu ký chứng chỉ bằng lệnh openssl req -new -nodes -newkey rsa:4096 -keyout key.pem -out cert.csr. Chúng tôi đã sử dụng các tùy chọn sau:

- req -new tạo yêu cầu ký chứng chỉ mới
- nodes lưu khóa riêng tư mà không có passphrase
- newkey tạo khóa riêng tư mới
- rsa:4096 tạo khóa RSA có kích thước 4096 bit
- keyout chỉ định nơi lưu trữ khóa
- out lưu yêu cầu ký chứng chỉ

Sau đó, bạn sẽ được yêu cầu trả lời một loạt các câu hỏi, như được hiển thị trong đầu ra console bên dưới.

```
user@TryHackMe$ openssl req -new -nodes -newkey rsa:4096 -keyout key.pem -out cert.csr
[...]
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:UK
State or Province Name (full name) []:London
Locality Name (eg, city) [Default City]:London
[...]
```

Sau khi tệp CSR đã sẵn sàng, bạn có thể gửi nó đến một CA của bạn để ký và sẵn sàng sử dụng trên máy chủ của bạn.

Khi máy khách, tức là trình duyệt, nhận được một chứng chỉ đã được ký mà nó tin tưởng, SSL/TLS handshake sẽ diễn ra. Mục đích là đồng ý về các cipher và khóa bí mật.

Chúng tôi vừa mô tả cách PKI áp dụng cho web và chứng chỉ SSL/TLS. Một bên thứ ba đáng tin cậy là cần thiết để hệ thống có thể mở rộng.

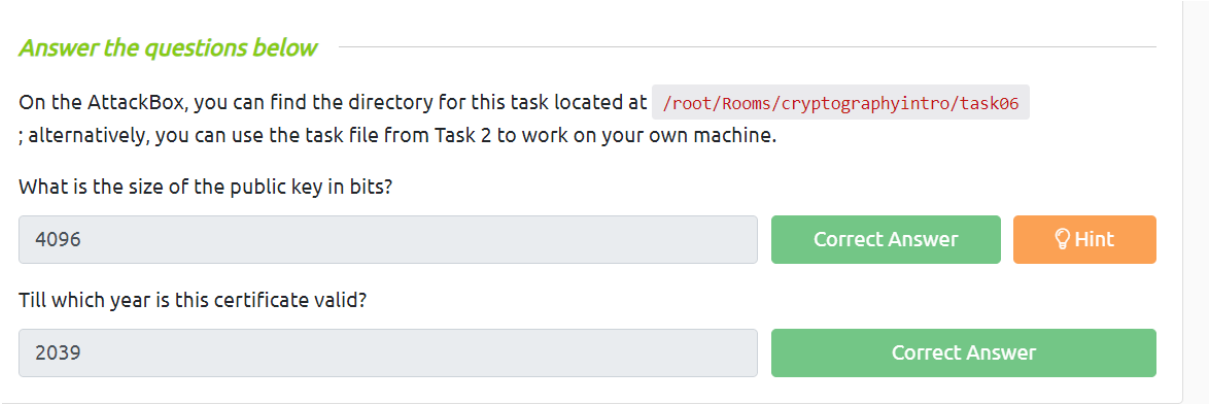
Cho mục đích kiểm tra, chúng tôi đã tạo một chứng chỉ tự ký. Ví dụ, lệnh sau sẽ tạo một chứng chỉ tự ký.

openssl req -x509 -newkey -nodes rsa:4096 -keyout key.pem -out cert.pem -sha256 -days 365

Tham số -x509 cho biết rằng chúng tôi muốn tạo một chứng chỉ tự ký thay vì yêu cầu chứng chỉ. Tham số -sha256 chỉ định việc sử dụng băm SHA-256. Nó sẽ có giá trị trong một năm vì chúng tôi đã thêm -days 365.

Để trả lời các câu hỏi bên dưới, bạn cần kiểm tra tệp chứng chỉ cert.pem trong thư mục task06. Bạn có thể sử dụng lệnh sau để xem chứng chỉ của mình:

openssl x509 -in cert.pem -text



VII. Authenticating with Passwords

Hãy xem cách mật mã hóa có thể giúp tăng cường bảo mật mật khẩu. Với PKI và SSL/TLS, chúng ta có thể giao tiếp với bất kỳ máy chủ nào và cung cấp thông tin đăng nhập của mình trong khi đảm bảo rằng không ai có thể đọc được mật khẩu của chúng ta khi chúng di chuyển trên mạng. Đây là một ví dụ về việc bảo vệ dữ liệu trong quá trình truyền. Hãy khám phá cách chúng ta có thể bảo vệ mật khẩu khi chúng được lưu trữ trong cơ sở dữ liệu, tức là dữ liệu yên tĩnh.

Phương pháp ít an toàn nhất là lưu tên người dùng và mật khẩu trong cơ sở dữ liệu. Như vậy, bất kỳ việc vi phạm dữ liệu nào đều sẽ tiết lộ mật khẩu của người dùng. Không cần nỗ lực nào ngoài việc đọc cơ sở dữ liệu chứa các mật khẩu.

Username	Password
alice	qwerty
bob	dragon
charlie	princess

Phương pháp cải thiện sẽ là lưu tên người dùng và phiên bản băm của mật khẩu trong cơ sở dữ liệu. Như vậy, một việc vi phạm dữ liệu sẽ tiết lộ các phiên bản băm của các mật khẩu. Vì hàm băm là không thể đảo ngược, kẻ tấn công cần phải tiếp tục thử các mật khẩu khác nhau để tìm ra mật khẩu nào sẽ dẫn đến cùng một băm. Bảng dưới đây cho thấy tổng MD5 của các mật khẩu. (Chúng tôi đã chọn MD5 chỉ để giữ trường mật khẩu nhỏ cho ví dụ; nếu không, chúng tôi sẽ sử dụng SHA256 hoặc một cái gì đó an toàn hơn.)

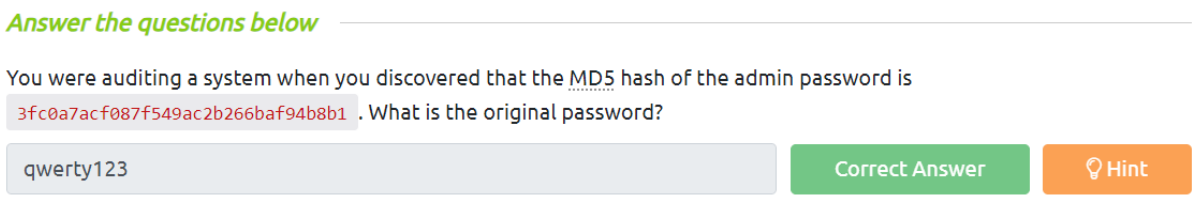
Username	Password

alice	5d41402abc4b2a76b9719d911017c592
bob	0dce7f1f6d6f8b9b8e7a9a5c8f1e1b76
charlie	3e23e8160039594a33894f6564e1b134

Bảng trên sử dụng hash(password + salt); một phương pháp khác sẽ là sử dụng hash(hash(password) + salt). Lưu ý rằng chúng tôi đã sử dụng một muối tương đối nhỏ cùng với hàm băm MD5. Chúng ta nên chuyển sang một hàm băm (an toàn hơn) và một muối lớn hơn để đảm bảo an toàn hơn nếu đây là một cài đặt thực tế.

Một cải tiến khác mà chúng ta có thể thực hiện trước khi lưu trữ mật khẩu là sử dụng một hàm tạo khóa như PBKDF2 (Password-Based Key Derivation Function 2). PBKDF2 lấy mật khẩu và muối và đưa nó qua một số lần lặp nhất định, thường là hàng trăm nghìn lần.

Chúng tôi khuyên bạn nên kiểm tra Bảng Các kỹ thuật lưu trữ mật khẩu nếu bạn muốn tìm hiểu về các kỹ thuật khác liên quan đến lưu trữ mật khẩu.



VIII. Cryptography and Data - Example

Trong nhiệm vụ này, chúng tôi muốn khám phá những gì xảy ra khi đăng nhập vào một trang web qua HTTPS.

1. Khách hàng yêu cầu chứng chỉ SSL/TLS của máy chủ
2. Máy chủ gửi chứng chỉ SSL/TLS cho khách hàng
3. Khách hàng xác nhận rằng chứng chỉ là hợp lệ

Vai trò của mật mã bắt đầu với việc kiểm tra chứng chỉ. Để chứng chỉ được coi là hợp lệ, điều đó có nghĩa là nó đã được ký. Việc ký chứng chỉ có nghĩa là một giá trị băm của chứng chỉ được mã hóa bằng khóa riêng tư của một bên thứ ba đáng tin cậy; giá trị băm được mã hóa được đính kèm vào chứng chỉ.

Nếu bên thứ ba được tin tưởng, khách hàng sẽ sử dụng khóa công khai của bên thứ ba để giải mã giá trị băm được mã hóa và so sánh nó với giá trị băm của chứng chỉ. Tuy nhiên, nếu bên thứ ba không được công nhận, kết nối sẽ không tiếp tục tự động.

Sau khi khách hàng xác nhận rằng chứng chỉ là hợp lệ, một SSL/TLS handshake được bắt đầu. Handshake này cho phép khách hàng và máy chủ đồng ý về khóa bí mật và thuật toán mã hóa đối xứng, ngoài những thứ khác. Từ điểm này trở đi, tất cả các phiên liên lạc liên quan sẽ được mã hóa bằng mã hóa đối xứng.

Bước cuối cùng sẽ là cung cấp thông tin đăng nhập. Khách hàng sử dụng phiên SSL/TLS được mã hóa để gửi chúng đến máy chủ. Máy chủ nhận tên người dùng và mật khẩu và cần xác nhận rằng chúng khớp.

Theo các hướng dẫn về bảo mật, chúng tôi mong đợi máy chủ lưu trữ phiên bản băm của mật khẩu sau khi đính kèm một muối ngẫu nhiên vào đó. Như vậy, nếu cơ sở dữ liệu bị xâm nhập, mật khẩu sẽ khó khôi phục.