MỤC LỤC

LỜI NÓI ĐẦU

Mật mã (Cryptography) là ngành khoa học là ngành nghiên cứu các kỹ thuật toán học nhằm cung cấp các dịch vụ bảo vệ thông tin. Đây là ngành khoa học quan trọng, có nhiều ứng dụng trong đời sống – xã hội.

Khoa học mật mã đã ra đời từ hàng nghìn năm. Tuy nhiên, trong suốt nhiều thế kỷ, các kết quả của lĩnh vực này hầu như không được ứng dụng trong các lĩnh vực dân sự thông thường của đời sống – xã hội mà chủ yếu được sử dụng trong lĩnh vực quân sự, chính trị, ngoại giao... Ngày nay, các ứng dụng mã hóa và bảo mật thông tin đang được sử dụng ngày càng phổ biến trong các lĩnh vực khác nhau trên thế giới, từ các lĩnh vực an ninh, quân sự, quốc phòng…, cho đến các lĩnh vực dân sự như thương mại điện tử, ngân hàng…

Với sự phát triển ngày càng nhanh chóng của Internet và các ứng dụng giao dịch điện tử trên mạng, nhu cầu bảo vệ thông tin trong các hệ thống và ứng dụng điện tử ngày càng được quan tâm và có ý nghĩa hết sức quan trọng. Các kết quả của khoa học mật mã ngày càng được triển khai trong nhiều lĩnh vực khác nhau của đời sống – xã hội, trong đó phải kể đến rất nhiều những ứng dụng đa dạng trong lĩnh vực dân sự, thương mại...Các ứng dụng mã hóa thông tin cá nhân, trao đổi thông tin kinh doanh, thực hiện các giao dịch điện tử qua mạng... đã trở nên gần gũi và quen thuộc với mọi người.

Cùng với sự phát triển của khoa học máy tính và Internet, các nghiên cứu và ứng dụng của mật mã học ngày càng trở nên đa dạng hơn, mở ra nhiều hướng nghiên cứu chuyên sâu vào từng lĩnh vực ứng dụng đặc thù với những đặc trưng riêng. Ứng dụng của khoa học mật mã không chỉ đơn thuần là mã hóa và giải mã thông tin mà còn bao gồm nhiều vấn đề khác nhau cần được nghiên cứu và giải quyết, ví dụ như chứng thực nguồn gốc nội dung thông tin (kỹ thuật chữ ký điện tử), chứng nhận tính xác thực về người sở hữu mã khóa (chứng nhận khóa công cộng), các quy trình giúp trao đổi thông tin và thực

hiện giao dịch điện tử an toàn trên mạng...

Các ứng dụng của mật mã học và khoa học bảo vệ thông tin rất đa dạng và phong phú, tùy vào tính đặc thù của mỗi hệ thống bảo vệ thông tin mà ứng dụng sẽ có các tính năng với đặc trưng riêng. Trong đó, chúng ta có thể kể ra một số tính năng chính của hệ thống bảo vệ thông tin:

- Tính bảo mật thông tin: hệ thống đảm bảo thông tin được giữ bí mật. Thông tin có thể bị phát hiện, ví dụ như trong quá trình truyền nhận, nhưng người tấn công không thể hiểu được nội dung thông tin bị đánh cắp này.
- Tính toàn vẹn thông tin: hệ thống bảo đảm tính toàn vẹn thông tin trong liên lạc hoặc giúp phát hiện rằng thông tin đã bị sửa đổi.
- Xác thực các đối tác trong liên lạc và xác thực nội dung thông tin trong liên lạc.
- Chống lại sự thoái thác trách nhiệm: hệ thống đảm bảo một đối tác bất kỳ trong hệ thống không thể từ chối trách nhiệm về hành động mà mình đã thực hiện.

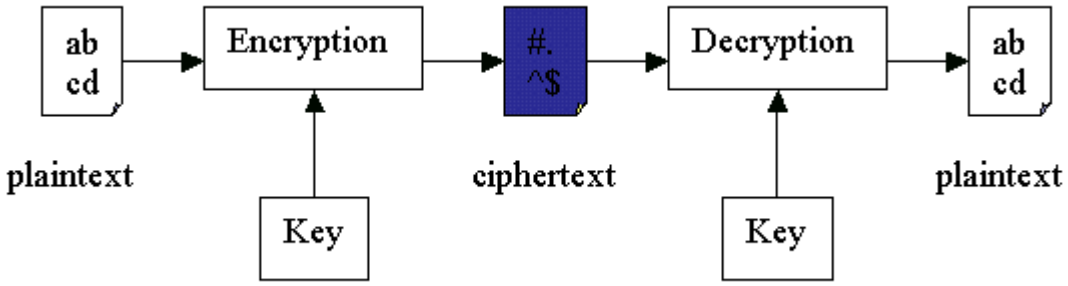
Những kết quả nghiên cứu về mật mã cũng đã được đưa vào trong các hệ thống phức tạp hơn, kết hợp với những kỹ thuật khác để đáp ứng yêu cầu đa dạng của các hệ thống ứng dụng khác nhau trong thực tế, ví dụ như hệ thống bỏ phiếu bầu cử qua mạng, hệ thống đào tạo từ xa, hệ thống quản lý an ninh của các đơn vị với hướng tiếp cận sinh trắc học, hệ thống cung cấp dịch vụ đa phương tiện trên mạng với yêu cầu cung cấp dịch vụ và bảo vệ bản quyền sở hữu trí tuệ đối với thông tin số...

Trong báo cáo thực tập này em sẽ trình bày quá trình tìm hiểu tổng quan về mã hóa và lớp Crytography trong .NET trong việc bảo vệ thông tin.

CHƯƠNG I: TỔNG QUAN VỀ MẬT MÃ HỌC

1.1 Mật mã học:

Mật mã học là ngành khoa học ứng dụng toán học vào việc biến đổi thông tin thành một dạng khác với mục đích che dấu nội dung, ý nghĩa thông tin cần mã hóa. Đây là một ngành quan trọng và có nhiều ứng dụng trong đời sống xã hội. Ngày nay, các ứng dụng mã hóa và bảo mật thông tin đang được sử dụng ngày càng phổ biến hơn trong các lĩnh vực khác nhau trên thế giới, từ các lĩnh vực an ninh, quân sự, quốc phòng…, cho đến các lĩnh vực dân sự như thương mại điện tử, ngân hàng…



Hình 1.1 : Sơ đồ mã hóa và giải mã

Cùng với sự phát triển của khoa học máy tính và Internet, các nghiên cứu và ứng dụng của khoa học mật mã ngày càng trở nên đa dạng hơn, mở ra nhiều hướng nghiên cứu chuyên sâu vào từng lĩnh vực ứng dụng đặc thù với những đặc trưng riêng. Ứng dụng của khoa học mật mã không chỉ đơn thuần là mã hóa và giải mã thông tin mà còn bao gồm nhiều vấn đề khác nhau cần được nghiên cứu và giải quyết: chứng thực nguồn gốc nội dung thông tin (kỹ thuật chữ ký điện tử), chứng nhận tính xác thực về người sở hữu mã khóa (chứng nhận khóa công cộng), các quy trình giúp trao đổi thông tin và thực hiện giao dịch điện tử an toàn trên mạng... Những kết quả nghiên cứu về mật mã cũng đã được đưa vào trong các hệ thống phức tạp hơn, kết hợp với những kỹ thuật khác để đáp ứng yêu cầu đa dạng của các hệ thống ứng dụng khác nhau trong thực tế, ví dụ như hệ thống bỏ phiếu bầu cử qua mạng, hệ thống đào tạo từ xa, hệ thống quản lý an ninh của các đơn vị với

hướng tiếp cận sinh trắc học, hệ thống cung cấp dịch vụ multimedia trên mạng với yêu cầu cung cấp dịch vụ và bảo vệ bản quyền sở hữu trí tuệ đối với thông tin số...

1.2 Hệ thống mã hóa :

- Định nghĩa 1.1: Hệ thống mã hóa (cryptosystem) là một bộ năm (P, C, K, E, D) thỏa mãn các điều kiện sau:

1. Tập nguồn P là tập hữu hạn tất cả các mẩu tin nguồn cần mã hóa có thể có.

2. Tập đích C là tập hữu hạn tất cả các mẩu tin có thể có sau khi mã hóa.

3. Tập khóa K là tập hữu hạn các khóa có thể được sử dụng.

4. E và D lần lượt là tập mã hóa và giải mã với mỗi khóa k ∈ K, tồn tại luật mã hóa $e_k \in E$ và luật giải mã $d_k \in D$ tương ứng. Luật mã hóa $e_k : P \rightarrow C$ và luật giải mã $d_k : C \rightarrow P$ là hai ánh xạ thỏa mãn $d_k(e_k(x)) = x \ \forall x \in P$.

Tính chất 4 là tính chất quan trọng của một hệ thống mã hóa. Tính chất này đảm bảo một mẩu tin $x \in P$ được mã hóa bằng luật mã hóa $e_k \in E$ có thể được giải mã chính xác bằng thuật giải mã $d_k \in D$.

- Định nghĩa 1.2: Z_m được định nghĩa là tập hợp $\{0,1,...,m-1\}$, được trang bị phép cộng (ký hiệu +) và phép nhân (ký hiệu là ×). Phép cộng và phép nhân trong Z_m được thực hiện tương tự như trong Z, ngoại trừ kết quả tính theo modulo m.

Ví dụ : Ta cần tính giá trị 11×13 trong Z_{16}. Trong Z, ta có kết quả phép nhân 11×13 = 143. Do 143 ≡ 15 (mod 16) nên 11×13 = 15 trong Z_{16}.

Một số tính chất của Z_m:

1. Phép cộng đóng trong $\mathbf{Z}_m$, $\forall a, b \in \mathbf{Z}_m$, $a + b \in \mathbf{Z}_m$
2. Tính giao hoán của phép cộng trong $\mathbf{Z}_m$, $\forall a, b \in \mathbf{Z}_m$, $a + b = b + a$
3. Tính kết hợp của phép cộng trong $\mathbf{Z}_m$, $\forall a, b, c \in \mathbf{Z}_m$, $(a + b) + c = a + (b + c)$
4. $\mathbf{Z}_m$ có phần tử trung hòa là 0, $\forall a, b \in \mathbf{Z}_m$, $a + 0 = 0 + a = a$
5. Mọi phần tử *a* trong $\mathbf{Z}_m$ đều có phần tử đối là $m - a$

6. Phép nhân đóng trong $\mathbf{Z}_m$, $\forall a,b \in \mathbf{Z}_m$, $a \times b \in \mathbf{Z}_m$

7. Tính giao hoán của phép nhân trong $\mathbf{Z}_m$, $\forall a,b \in \mathbf{Z}_m$, $a \times b = b \times a$

8. Tính kết hợp của phép nhân trong $\mathbf{Z}_m, \forall a,b,c \in \mathbf{Z}_m$, $(a \times b) \times c = a \times (b \times c)$

9. $\mathbf{Z}_m$ có phần tử đơn vị là 1, $\forall a,b \in \mathbf{Z}_m$, $a \times 1 = 1 \times a = a$

10. Tính chất phân phối của phép nhân đối với phép cộng $\forall a,b,c \in \mathbf{Z}_m$, $(a+b) \times c = a \times c + b \times c$

1.3 Các tính chất cơ bản của quá trình bảo mật và mã hóa:

- Tính bí mật (confidentiality/privacy): tính chất này đảm bảo thông tin chỉ được hiểu bởi những ai biết chìa khóa bí mật.

- Tính toàn vẹn (integrity): tính chất này đảm bảo thông tin không thể bị thay đổi mà không bị phát hiện. Tính chất này không đảm bảo thông tin không bị thay đổi, nhưng một khi nó bị nghe lén hoặc thay đổi thì người nhận được thông tin có thể biết được là thông tin đã bị nghe lén hoặc thay đổi. Các hàm một chiều (one-way function) như MD5, SHA-1, MAC...được dùng để đảm bảo tính toàn vẹn cho thông tin.

- Tính xác thực (authentication): người gửi (hoặc người nhận) có thể chứng minh đúng họ. Người ta có thể dụng một password, một challenge dựa trên một thuật toán mã hóa hoặc một bí mật chia sẻ giữa hai người để xác thực. Sự xác thực này có thể thực hiện một chiều (one-way) hoặc hai chiều (multual authentication).

- Tính không chối bỏ (non-repudiation): người gửi hoặc nhận sau này không thể chối bỏ việc đã gửi hoặc nhận thông tin. Thông thường điều này được thực hiện thông qua một chữ ký điện tử (electronic signature).

- Tính nhận dạng (identification): người dùng của một hệ thống, một tài nguyên sở hữu một chứng minh thư (identity) như là một chìa khóa ban đầu (primary key). identity này sẽ xác định những chức năng của người dùng, giới hạn cho phép của người dùng cũng như các thuộc tính liên quan (thường

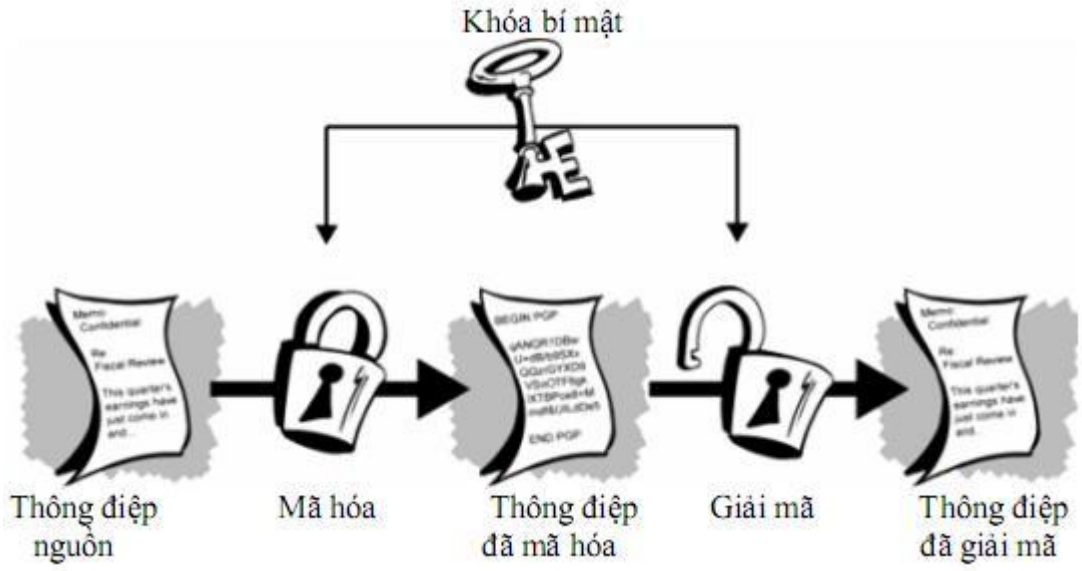
gọi chung là credential). Identity có thể là login, dấu vân tay, ADN, giản đồ võng mạc mắt…

1.4 Hệ thống mã hóa quy ước (mã hóa đối xứng) :

Trong hệ thống mã hóa quy ước, quá trình mã hóa và giải mã một thông điệp sử dụng cùng một mã khóa gọi là khóa bí mật (secret key) hay khóa đối xứng (symmetric key). Do đó, vấn đề bảo mật thông tin đã mã hóa hoàn toàn phụ thuộc vào việc giữ bí mật nội dung của mã khóa đã được sử dụng.

Với tốc độ và khả năng xử lý ngày càng được nâng cao của các bộ vi xử lý hiện nay, phương pháp mã hóa chuẩn (Data Encryption Standard – DES) đã trở nên không an toàn trong bảo mật thông tin. Do đó, Viện Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ (National Institute of Standards and Technology – NIST) đã quyết định chọn một chuẩn mã hóa mới với độ an toàn cao nhằm phục vụ nhu cầu bảo mật thông tin liên lạc của chính phủ Hoa Kỳ cũng như trong các ứng dụng dân sự. Thuật toán Rijndael do Vincent Rijmen và Joan Daeman đã được chính thức chọn trở thành chuẩn mã hóa nâng cao (Advanced Encryption Standard –AES) từ 02 tháng 10 năm 2000.

Ví dụ thông điệp nguồn được mã hóa với mã khóa k được thống nhất trước giữa người gửi A và người nhận B. Người A sẽ sử dụng mã khóa k để mã hóa thông điệp x thành thông điệp y và gửi y cho người B người B sẽ sử dụng mã khóa k để giải mã thông điệp y này. Vấn đề an toàn bảo mật thông tin được mã hóa phụ thuộc vào việc giữ bí mật nội dung mã khóa k.Nếu người C biết được mã khóa k thì C có thể “mở khóa” thông điệp đã được mã hóa mà người A gửi cho người B.



Hình 1.2 : Mô hình hệ thống mã hóa qui ước

1.4.1 Phương pháp mã hóa dịch chuyển :

Phương pháp mã hóa dịch chuyển là một trong những phương pháp lâu đời nhất được sử dụng để mã hóa. Thông điệp được mã hóa bằng cách dịch chuyển xoay vòng từng ký tự đi k vị trí trong bảng chữ cái.

Trong trường hợp đặc biệt k=3, phương pháp mã hóa dịch chuyển được gọi là phương pháp mã hóa Caesar.

Thuật toán 1.1: Phương pháp mã hóa dịch chuyển

Cho $P = C = K = \mathbb{Z}_n$

Với mỗi khóa $k \in K$, định nghĩa:

$e_k(x) = (x + k) \bmod n$ và $d_k(y) = (y - k) \bmod n$ với $x, y \in \mathbb{Z}_n$

$E = \{e_k, k \in K\}$ và $D = \{d_k, k \in K\}$

Mã hóa dịch chuyển là một phương pháp mã hóa đơn giản, thao tác xử lý mã hóa và giải mã được thực hiện nhanh chóng. Tuy nhiên, trên thực tế, phương pháp này có thể dễ dàng bị phá vỡ bằng cách thử mọi khả năng khóa $k \in K$. Điều này hoàn toàn có thể thực hiện được do không gian khóa K chỉ có n phần tử để chọn lựa.

Ví dụ: Để mã hóa một thông điệp được biểu diễn bằng các chữ cái từ A đến Z (26 chữ cái), ta sử dụng P = C = K = Z26. Khi đó, thông điệp được mã hóa sẽ không an toàn và có thể dễ dàng bị giải mã bằng cách thử lần lượt 26 giá trị khóa k € K. Tính trung bình, thông điệp đã được mã hóa có thể bị giải mã sau khoảng *n/2* lần thử khóa *k € K*.

1.4.2 Phương pháp mã hóa thay thế :

Phương pháp mã hóa thay thế (Substitution Cipher) là một trong những phương pháp mã hóa nổi tiếng và đã được sử dụng từ hàng trăm năm nay. Phương pháp này thực hiện việc mã hóa thông điệp bằng cách hoán vị các phần tử trong bảng chữ cái hay tổng quát hơn là hoán vị các phần tử trong tập nguồn P.

Thuật toán 1.2: Phương pháp mã hóa thay thế

Cho $P = C = Z_n$

K là tập hợp tất cả các hoán vị của n phần tử $0, 1, ..., n-1$. Như vậy, mỗi khóa $\pi \in K$ là một hoán vị của n phần tử $0, 1, ..., n-1$.

Với mỗi khóa $\pi \in K$, định nghĩa:

$e_\pi(x) = \pi(x)$ và $d_\pi(y) = \pi^{-1}(y)$ với $x, y \in \mathbb{Z}_n$

$E = \{e_\pi, \pi \in K\}$ và $D = \{D_\pi, \pi \in K\}$

Đây là một phương pháp đơn giản, thao tác mã hóa và giải mã được thực hiện nhanh chóng. Phương pháp này khắc phục điểm hạn chế của phương pháp mã

hóa bằng dịch chuyển là có không gian khóa K nhỏ nên dễ dàng bị giải mã bằng cách thử nghiệm lần lượt n giá trị khóa *k € K*. Trong phương pháp mã hóa thay thế có không gian khóa K rất lớn với *n*! phần tử nên không thể bị giải mã bằng cách "vét cạn" mọi trường hợp khóa k. Tuy nhiên, trên thực tế thông điệp được mã hóa bằng phương pháp này vẫn có thể bị giải mã nếu như có thể thiết lập được bảng tần số xuất hiện của các ký tự trong thông điệp hay nắm được một số từ, ngữ trong thông điệp nguồn ban đầu.

1.4.3 Phương pháp Affine :

Nếu như phương pháp mã hóa bằng dịch chuyển là một trường hợp đặc biệt của phương pháp mã hóa bằng thay thế, trong đó chỉ sử dụng *n* giá trị khóa *k* trong số *n*! phần tử, thì phương pháp Affine lại là một trường hợp đặc biệt khác của mã hóa bằng thay thế.

Thuật toán 1.3: Phương pháp Affine

Cho $P = C = Z_n$

$K = \{(a,b) \in \mathbb{Z}_n \times \mathbb{Z}_n : \gcd(a,n) = 1\}$

Với mỗi khóa $k = (a,b) \in K$, định nghĩa:

$e_k(x) = (ax+b) \bmod n$ và $d_k(x) = (a^{-1}(y-b)) \bmod n$ với $x, y \in \mathbb{Z}_n$

$E = \{e_k, k \in K\}$ và $D = \{D_k, k \in K\}$

1.4.4 Phương pháp Vigenere :

Trong phương pháp mã hóa bằng thay thế cũng như các trường hợp đặc biệt của phương pháp này (mã hóa bằng dịch chuyển, mã hóa Affine,…), ứng với một khóa *k* được chọn, mỗi phần tử $x \in P$ được ánh xạ vào duy nhất một phần tử $y \in C$. Nói cách khác, ứng với mỗi khóa $k \in K$, một song ánh được thiết lập từ P vào C.

Khác với hướng tiếp cận này, phương pháp Vigenere sử dụng một từ khóa có độ dài *m*. Có thể xem như phương pháp mã hóa Vigenere Cipher bao gồm *m* phép mã hóa bằng dịch chuyển được áp dụng luân phiên nhau theo chu kỳ.

Không gian khóa *K* của phương pháp Vigenere Cipher có số phần tử là n^m, lớn hơn hẳn phương pháp số lượng phần tử của không gian khóa *K* trong phương pháp mã hóa bằng dịch chuyển. Do đó, việc tìm ra mã khóa *k* để giải mã thông điệp đã được mã hóa sẽ khó khăn hơn đối với phương pháp mã hóa bằng dịch chuyển.

Thuật toán 1.4: Phương pháp mã hóa Vigenere

Chọn số nguyên dương m. Định nghĩa $P = C = K = (\mathbb{Z}_n)^m$

$K = \{(k_0, k_1, ..., k_{r-1}) \in (\mathbb{Z}_n)^r\}$

Với mỗi khóa $k = (k_0, k_1, ..., k_{r-1}) \in K$, định nghĩa:

$e_k(x_1, x_2, ..., x_m) = ((x_1 + k_1) \bmod n, (x_2 + k_2) \bmod n, ..., (x_m + k_m) \bmod n)$

$d_k(y_1, y_2, ..., y_m) = ((y_1 - k_1) \bmod n, (y_2 - k_2) \bmod n, ..., (y_m - k_m) \bmod n)$

với $x, y \in (\mathbb{Z}_n)^m$.

1.4.5 Phương pháp Hill :

Phương pháp Hill được Lester S. Hill công bố năm 1929: Cho số nguyên dương *m*, định nghĩa P = C = $(Z_n)^m$. Mỗi phần tử $x \in P$ là một bộ *m* thành phần, mỗi thành phần thuộc Z_n. Ý tưởng chính của phương pháp này là sử dụng *m* tổ hợp tuyến tính của *m* thành phần trong mỗi phần tử $x \in P$ để phát sinh ra *m* thành phần tạo thành phần tử $y \in C$.

Thuật toán 1.5: Phương pháp mã hóa Hill

Chọn số nguyên dương m. Định nghĩa:

$P = C = (\mathbb{Z}_n)^m$ và K là tập hợp các ma trận $m \times m$ khả nghịch

Với mỗi khóa $k = \begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,m} \\ k_{2,1} & \cdots & \cdots & k_{2,m} \\ \vdots & \vdots & & \vdots \\ k_{m,1} & k_{m,2} & \cdots & k_{m,m} \end{pmatrix} \in K$, định nghĩa:

$$e_k(x) = xk = (x_1, x_2, ..., x_m) \begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,m} \\ k_{2,1} & \cdots & \cdots & k_{2,m} \\ \vdots & \vdots & & \vdots \\ k_{m,1} & k_{m,2} & \cdots & k_{m,m} \end{pmatrix} \text{ với } x = (x_1, x_2, ..., x_m) \in P$$

và $d_k(y) = yk^{-1}$ với $y \in C$.

Mọi phép toán số học đều được thực hiện trên $\mathbb{Z}_n$.

1.4.6 Phương pháp mã hóa hoán vị :

Những phương pháp mã hóa nêu trên đều dựa trên ý tưởng chung: thay thế mỗi ký tự trong thông điệp nguồn bằng một ký tự khác để tạo thành thông điệp đã được mã hóa. Ý tưởng chính của phương pháp mã hóa hoán vị (Permutation Cipher) là vẫn giữ nguyên các ký tự trong thông điệp nguồn mà chỉ thay đổi vị trí các ký tự; nói cách khác thông điệp nguồn được mã hóa bằng cách sắp xếp lại các ký tự trong đó.

Thuật toán 1.6: Phương pháp mã hóa hoán vị

Chọn số nguyên dương m. Định nghĩa:

$P = C = (\mathbb{Z}_n)^m$ và K là tập hợp các hoán vị của m phần tử $\{1, 2, ..., m\}$

Với mỗi khóa $\pi \in K$, định nghĩa:

$e_\pi(x_1, x_2, ..., x_m) = (x_{\pi(1)}, x_{\pi(2)}, ..., x_{\pi(m)})$ và

$d_\pi(y_1, y_2, ..., y_m) = (y_{\pi^{-1}(1)}, y_{\pi^{-1}(2)}, ..., y_{\pi^{-1}(m)})$

với π^{-1} hoán vị ngược của π

Phương pháp mã hóa bằng hoán vị chính là một trường hợp đặc biệt của phương pháp Hill. Với mỗi hoán vị π của tập hợp {1, 2, ..., m} , ta xác định ma trận $k_\pi=(k_{i,j})$ theo công thức sau:

- $k_{i,j} = 1$ *nếu* $i = \pi(j)$
- $k_{i,j} = 1$ *nếu 0 trong trường hợp ngược lại*

Ma trận k_π là ma trận mà mỗi dòng và mỗi cột có đúng một phần tử mang giá trị 1, các phần tử còn lại trong ma trận đều bằng 0. Ma trận này có thể thu được bằng cách hoán vị các hàng hay các cột của ma trận đơn vị I_m nên k_π là ma trận khả nghịch. Rõ ràng, mã hóa bằng phương pháp Hill với ma trận k_π hoàn toàn tương đương với mã hóa bằng phương pháp hoán vị với hoán vị π.

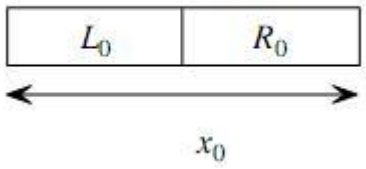
1.4.7 Phương pháp DES (Data Encryption Standard):

Khoảng những năm 1970, tiến sĩ Horst Feistel đã đặt nền móng đầu tiên cho chuẩn mã hóa dữ liệu DES với phương pháp mã hóa Feistel Cipher. Vào năm 1976 Cơ quan Bảo mật Quốc gia Hoa Kỳ (NSA) đã công nhận DES dựa trên phương pháp Feistel là chuẩn mã hóa dữ liệu. Kích thước khóa của DES ban đầu là 128 bit nhưng tại bản công bố FIPS kích thước khóa được rút xuống còn 56 bit.

Trong phương pháp DES, kích thước khối là 64 bit. DES thực hiện mã hóa dữ liệu qua 16 vòng lặp mã hóa, mỗi vòng sử dụng một khóa chu kỳ 48 bit được tạo ra từ khóa ban đầu có độ dài 56 bit. DES sử dụng 8 bảng hằng số S-box để thao tác.

Quá trình mã hóa của DES có thể được tóm tắt như sau: Biểu diễn thông điệp nguồn $x \in P$ bằng dãy 64bit. Khóa k có 56 bit. Thực hiện mã hóa theo ba giai đoạn:

1. Tạo dãy 64 bit x_0 bằng cách hoán vị x theo hoán vị IP (Initial Permutation). Biểu diễn $x_0 = IP(x) = L_0 R_0$, L_0 gồm 32 bit bên trái của x_0, R_0 gồm 32 bit bên phải của x_0.



Hình 1.3 : Biểu diễn 64 bit x thành 2 phần L và R

2. Thực hiện 16 vòng lặp từ 64 bit thu được và 56 bit của khoá k (chỉ sử dụng 48 bit của khoá k trong mỗi vòng lặp). 64 bit kết quả thu được qua mỗi vòng lặp sẽ là đầu vào cho vòng lặp sau. Các cặp từ 32 bit L_i, R_i (với $1 \leq i \leq 16$) được xác định theo quy tắc sau:

$L_i = R_{i-1}$

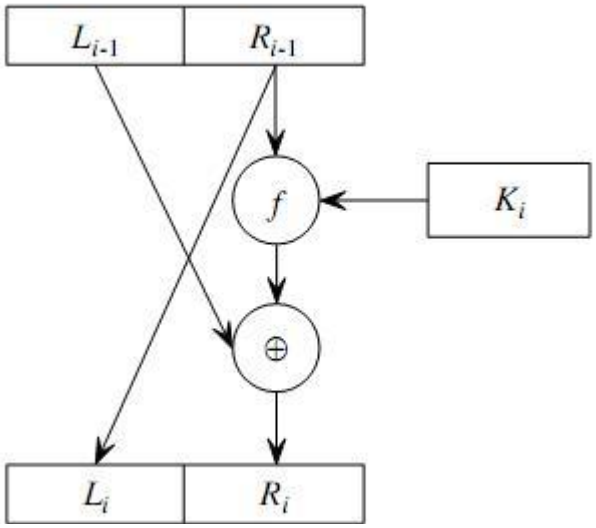
$R_i = L_{i-1}$ *XOR* $f(R_{i-1}, K_i)$

XOR trên hai dãy bit, K_1, K_2, ..., K_{16} là các dãy 48 bit phát sinh từ khóa K cho trước (Trên thực tế, mỗi khóa K_i được phát sinh bằng cách hoán vị các bit trong khóa K cho trước).

3. Áp dụng hoán vị ngược IP^{-1} đối với dãy bit $R_{16}L_{16}$, thu được y gồm 64 bit. Như vậy, $y = IP^{-1}(R_{16}L_{16})$.

Hàm f được sử dụng ở bước 2 là hàm có gồm hai tham số: Tham số thứ nhất A là một dãy 32 bit, tham số thứ hai J là một dãy 48 bit. Kết quả của hàm f là một dãy 32 bit. Các bước xử lý của hàm f(A , J) như sau:

Tham số thứ nhất A (32 bit) được mở rộng thành dãy 48 bit bằng hàm mở rộng E. Kết quả của hàm $E(A)$ là một dãy 48 bit được phát sinh từ A bằng cách hoán vị theo một thứ tự nhất định 32 bit của A, trong đó có 16 bit của A được lặp lại hai lần trong $E(A)$.



Hình 1.4 : Qui trình phát sinh dãy L_i, R_i từ dãy L_{i-1}, R_{i-1} và khóa K_i

Thực hiện phép toán XOR cho hai dãy 48 bit $E(A)$ và J, ta thu được một dãy 48 bit B. Biểu diễn B thành từng nhóm 6 bit như sau: $B = B_1B_2B_3B_4B_5B_6B_7B_8$.

Sử dụng tám ma trận S_1, S_2,..., S_8, mỗi ma trận S_i có kích thước 4x16 và mỗi dòng của ma trận nhận đủ 16 giá trị từ 0 đến 15. Xét dãy gồm 6 bit $B_i = b_1b_2b_3b_4b_5b_6$, $S_j(B_j)$ được xác định bằng giá trị của phần tử tại dòng r cột c của

S_j, trong đó, chỉ số dòng r có biểu diễn nhị phân là b_1b_6, chỉ số cột c có biểu diễn nhị phân là $b_1b_2b_4b_5$. Bằng cách này, ta xác định được dãy 4 bit $C_j = S_j(B_j)$, $1 \leq j \leq 8$.

Tập hợp các dãy 4 bit C_j lại, ta có được dãy 32 bit $C=C_1C_2C_3C_4C_5C_6C_7C_8$. Dãy 32 bit thu được bằng cách hoán vị C theo một quy luật P nhất định chính là kết quả của hàm $F(A,J)$.

Quá trình giải mã chính là thực hiện theo thứ tự đảo ngược các thao tác của quá trình mã hóa.

1.4.8 Phương pháp mã hóa Rijndael:

Với tốc độ và khả năng xử lý ngày càng được nâng cao của các bộ vi xử lý hiện nay, phương pháp mã hóa chuẩn (Data Encryption Standard – DES) trở nên không an toàn trong bảo mật thông tin. Do đó, Viện Tiêu chuẩn và Công nghệ Hoa Kỳ (National Institute of Standards and Technology – NIST) đã quyết định chọn một chuẩn mã hóa mới với độ an toàn cao nhằm phục vụ nhu cầu bảo mật thông tin liên lạc của Chính phủ Hoa Kỳ cũng như trong các ứng dụng dân sự. Thuật toán Rijndael do Vincent Rijmen và Joan Daeman đã được chính thức chọn trở thành chuẩn mã hóa nâng cao AES (Advanced Encryption Standard) từ ngày 02 tháng 10 năm 2000.

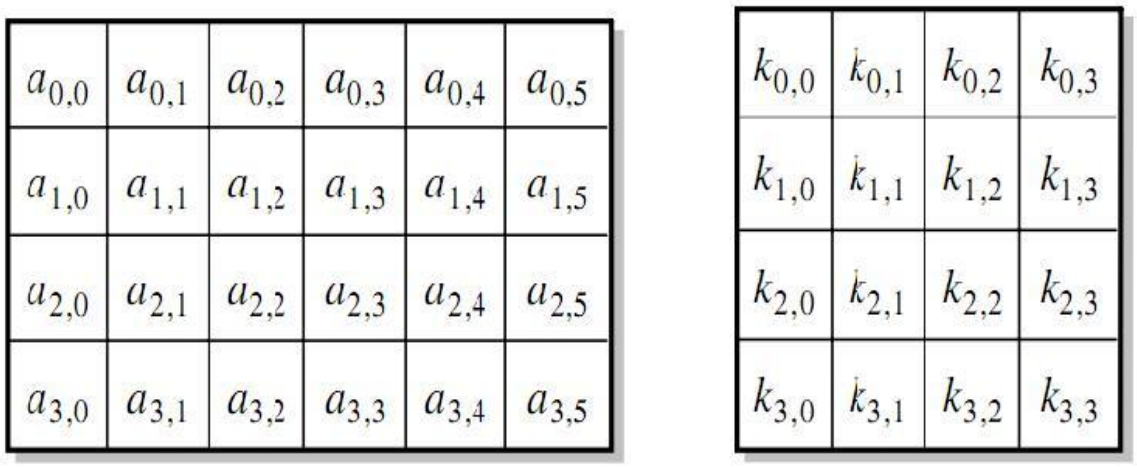
Phương pháp mã hóa Rijndael là phương pháp mã hóa theo khối (block cipher)có kích thước khối và mã khóa thay đổi linh hoạt với các giá trị 128, 192 hay 256 bit. Phương pháp này thích hợp ứng dụng trên nhiều hệ thống khác nhau từ các thẻ thông minh cho đến các máy tính cá nhân.

Phương pháp mã hóa Rijndael bao gồm nhiều bước biến đổi được thực hiện tuần tự, kết quả đầu ra của bước biến đổi trước là đầu vào của bước biến đổi tiếp theo. Kết quả trung gian giữa các bước biến đổi được gọi là trạng thái (state).

Một trạng thái có thể được biểu diễn dưới dạng một ma trận gồm 4 dòng và Nb cột với Nb bằng với độ dài của khối chia cho 32. Mã khóa chính (Cipher Key) cũng được biểu diễn dưới dạng một ma trận gồm 4 dòng và Nk

cột với Nk bằng với độ dài của khóa chia cho 32. Trong một số tình huống, ma trận biểu diễn một trạng thái hay mã khóa có thể được khảo sát như mảng một chiều chứa các phần tử có độ dài 4 byte, mỗi phần tử tương ứng với một cột của ma trận.

Số lượng chu kỳ, ký hiệu là Nr, phụ thuộc vào giá trị của Nb và Nk theo công thức: Nr = max{Nb, Nk} + 6.



Hình 1.5 : Biểu diễn ma trận trạng thái (Nb = 6) và khóa (Nk = 4)

Quy trình mã hóa Rijndael sử dụng bốn phép biến đổi chính:

1. AddRoundKey: cộng mã khóa của chu kỳ vào trạng thái hiện hành. Độ dài của mã khóa của chu kỳ bằng với kích thước của trạng thái.
2. SubBytes: thay thế phi tuyến mỗi byte trong trạng thái hiện hành thông qua bảng thay thế (S-box).
3. MixColumns: trộn thông tin của từng cột trong trạng thái hiện hành. Mỗi cột được xử lý độc lập.
4. ShiftRows: dịch chuyển xoay vòng từng dòng của trạng thái hiện hành với di số khác nhau.

Mỗi phép biến đổi thao tác trên trạng thái hiện hành S. Kết quả S' của mỗi phép biến đổi sẽ trở thành đầu vào của phép biến đổi kế tiếp trong quy trình mã hóa.

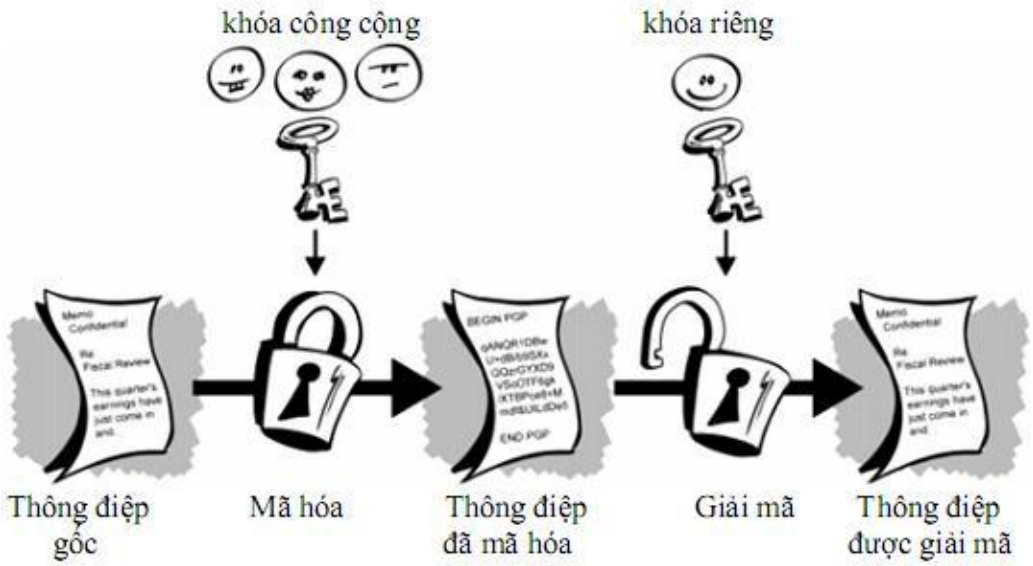
Với các hàm:

AddRoundKey	Phép biến đổi sử dụng trong mã hóa và giải mã, thực hiện việc cộng mã khóa của chu kỳ vào trạng thái hiện hành. Độ dài của mã khóa của chu kỳ bằng với kích thước của trạng thái.
SubBytes	Phép biến đổi sử dụng trong mã hóa,thực hành việc thay thế phi tuyến từngbyte trong trạng thái hiện hành thông qua bảng thay thế (S-box).
MixColumns	Phép biến đổi sử dụng trong mã hóa, thực hiện thao tác trộn thông tin của từng cột trong trạng thái hiện hành. Mỗi cột được xử lý độc lập.
ShiftRows	Phép biến đổi sử dụng trong mã hóa, thực hiện việc dịch chuyển xoay vòng từng dòng của trạng thái hiện hành với di số tương ứng khác nhau.

1.5 Hệ thống mã hóa khóa công cộng (mã hóa bất đối xứng) và phương pháp RSA :

Nếu như vấn đề khó khăn đặt ra đối với các phương pháp mã hóa quy ước chính là bài toán trao đổi mã khóa thì ngược lại, các phương pháp mã hóa khóa công cộng giúp cho việc trao đổi mã khóa trở nên dễ dàng hơn. Nội dung của khóa công cộng (public key) không cần phải giữ bí mật như đối với khóa bí mật trong các phương pháp mã hóa quy ước. Sử dụng khóa công cộng, chúng ta có thể thiết lập một quy trình an toàn để truy đổi khóa bí mật được sử dụng trong hệ thống mã hóa quy ước.

Trong những năm gần đây, các phương pháp mã hóa khóa công cộng, đặc biệt là phương pháp RSA [45], được sử dụng ngày càng nhiều trong các ứng dụng mã hóa trên thế giới và có thể xem như đây là phương pháp chuẩn được sử dụng phổ biến nhất trên Internet, ứng dụng trong việc bảo mật thông tin liên lạc cũng như trong lĩnh vực thương mại điện tử.



Hình 1.3 : Mô hình hệ thống mã hóa với khóa công cộng

Năm 1978, R.L.Rivest, A.Shamir và L.Adleman đã đề xuất hệ thống mã hóa khóa công cộng RSA (hay còn được gọi là "hệ thống MIT"). Trong phương pháp này, tất cả các phép tính đều được thực hiện trên Z_n với n là tích của hai số nguyên tố lẻ p và q khác nhau. Khi đó, ta có $\Phi(n) = (p–1)(q–1)$.

Thuật toán 1.8: Phương pháp RSA

$n = pq$ với p và q là hai số nguyên tố lẻ phân biệt.
Cho $P = C = \mathbb{Z}_n$ và định nghĩa:
$K = \{((n, p, q, a, b): n = pq, p, q$ là số nguyên tố, $ab \equiv 1 \pmod{\phi(n)}\}$
Với mỗi $k = (n, p, q, a, b) \in K$, định nghĩa:
$e_k(x) = x^b \bmod n$ và $d_k(y) = y^a \bmod n$, với $x, y \in \mathbb{Z}_n$
Giá trị n và b được công bố, trong khi giá trị p, q, a được giữ bí mật

Dựa trên định nghĩa phương pháp mã hóa RSA, việc áp dụng vào thực tế được tiến hành theo các bước sau:

Thuật toán 1.9: Sử dụng phương pháp RSA

Phát sinh hai số nguyên tố có giá trị lớn p và q
Tính $n = pq$ và $\phi(n) = (p - 1)(q - 1)$
Chọn ngẫu nhiên một số nguyên b $(1 < b < \phi(n))$ thỏa $\gcd(b, \phi(n)) = 1$
Tính giá trị $a = b^{-1} \bmod \phi(n)$ (bằng thuật toán Euclide mở rộng)
Giá trị n và b được công bố (khóa công cộng), trong khi giá trị p, q, a được giữ bí mật (khóa riêng)

CHƯƠNG II: .NET FRAMEWORK

2.1 Định nghĩa .NET :

Để bắt đầu công việc khảo sát .NET, chúng ta phải xác định rõ ranh giới công việc cần thực hiện. Vì .NET có nghĩa một nền tảng hơn là một sản phẩm đơn lẻ, cho nên cách định nghĩa nó có thể đa dạng, có phần hơi khó hiểu và mơ hồ. Một cách đơn giản .NET được định nghĩa dưới dạng một khung ứng dụng (application framework). .NET cung cấp một khung cho những ứng dụng nào được xây dựng; nó xác định những ứng dụng truy nhập các hàm như thế nào qua các hệ thống và các mạng. .Net cung cấp một nền tảng mà trên đó các giải pháp và các dịch vụ Web có thể được xây dựng, một nền tảng giải phóng những sự ràng buộc và tự bản thân nó giải phóng khỏi Microsoft Windows (về mặt kĩ thuật). Nói cách khác, .NET là một cách để xây dựng các ứng dụng và các dịch vụ mà nó hoạt động không phụ thuộc vào một nền tảng (platform) nào. Đây là một cách để tạo ra các trao đổi thông tin (truyền thông) giữa những hệ thống đa dạng và các ứng dụng cũng như tích hợp nhiều thiết bị vào trong việc trao đổi thông tin này.

Ý tưởng .NET được thiết kế để hỗ trợ chúng ta tiến tới một Web thân thiện hơn, tích hợp tốt hơn, một nơi mà ở đó các ứng dụng và các quá trình giao dịch có thể tương tác với nhau một cách tự do không phụ thuộc vào chương trình và nền tảng. Tóm lại, .NET làm cho thông tin trên Web có thể được tiếp cận một cách dễ dàng: bạn có thể sử dụng bất kì thiết bị nào, trên bất kì nền tảng nào. .NET còn có thể hỗ trợ các hệ thống máy phục vụ và ứng dụng liên lạc với nhau một cách thông suốt (seamlessly) và xây dựng hệ thống tính toán phân tán trên Web, làm cho Web trở thành một nơi tương tác năng động hơn giữa các dịch vụ Web, các ứng dụng và khách hàng.

2.2 Mục tiêu của .NET :

Microsoft .NET trợ giúp loại bỏ các thành phần riêng biệt khỏi một nền tảng và ứng dụng và như vậy nó cho phép thông tin được trao đổi và xây dựng trên một nền tảng chung hơn. Bạn có thể nghĩ "Microsoft được lợi gì từ ý

tưởng này?". Xét cho cùng, việc tạo ra một nền tảng (platform) độc lập và nó không cần các sản phẩm của Microsoft để thực thi (implement) xem ra đã tự phá huỷ. Trên thực tế Microsoft đang ôm một ý tưởng ở đâu đó và tại một lúc nào đó, các nhà phát triển công nghệ phải đua tranh với nhau ở mức ứng dụng và dịch vụ chứ không phải là mức nền tảng (platform level).

Hãy xem một ví dụ: Nếu bạn vào một cửa hàng bách hoá để mua bóng đèn, bạn sẽ nhìn thấy nhiều loại bóng đèn khác nhau được làm bởi các nhà sản xuất khác nhau. Song, bóng đèn bạn mua hoàn toàn phù hợp với những tiêu chuẩn nhất định. Nói cách khác, bất chấp nhãn hiệu, độ bền của bóng đèn và khả năng tương thích của bóng với đui đèn là tương đương nhau. Vì đây là ngành công nghiệp bóng đèn cạnh tranh với nhau ở mức sản phẩm, chứ không phải ở mức các chuẩn. Nếu mỗi một công ty đua tranh ở mức các chuẩn, bạn sẽ có các sản phẩm bóng đèn khác nhau mà chúng chỉ được sản xuất độc quyền cho các kiểu đui đèn riêng biệt, và đây là một kế hoạch chắc chắn làm cho việc mua hàng trở thành một sự thách đố cho khách hàng.

.NET Framework là thành quả tối ưu của sự kết hợp công sức và trí tuệ của Microsoft, nhằm tạo ra một nền tảng cho việc xây dựng và triển khai nhanh chóng các dịch vụ và ứng dụng Web XML. Tầm nhìn của nền tảng .NET Framework kết hợp một mô hình lập trình đơn giản, dễ sử dụng với các giao thức mở và biến đổi được của Internet. Để đạt được tầm nhìn này, việc thiết kế .NET Framework nhằm một số mục đích:

- ***Sự hợp nhất thông qua các chuẩn Internet công cộng*:** Để giao tiếp với những đối tác kinh doanh, những khách hàng phụ thuộc vào các khu vực theo vị trí địa lý, thậm trí cả những ứng dụng cho tương lai, những giải pháp phát triển cần được đề nghị hỗ trợ cho các chuẩn Internet mở và tích hợp chặt chẽ với các giao thức mà không bắt buộc người phát triển phải thông hiểu cơ sở hạ tầng bên dưới nó.

- ***Khả năng biến đổi được thông qua một kiến trúc "ghép nối lỏng":*** Đa số các hệ thống lớn, biến đổi được trên thế giới được xây dựng trên những

kiến trúc không đồng bộ dựa trên nền thông điệp (message-based). Nhưng công việc xây dựng các ứng dụng trên một kiến trúc như vậy thường phức tạp và có ít các công cụ hơn so với những môi trường phát triển ứng dụng N lớp (N-tier) "*ghép nối chặt*". .NET Framework được xây dựng để đem lại những lợi thế về năng suất của kiến trúc "*ghép nối chặt"* với khả năng biến đổi được và vận hành với nhau của kiến trúc "*ghép nối lỏng"*.

- ***Hỗ trợ nhiều ngôn ngữ:*** Các nhà phát triển sử dụng những ngôn ngữ khác nhau do mỗi ngôn ngữ riêng có những ưu thế đặc thù : một số ngôn ngữ đặc biệt thích hợp với thao tác toán học; một số khác lại đa dạng ở các hàm tính toán tài chính v.v. .NET Framework cho phép các ứng dụng được viết trong nhiều ngôn ngữ lập trình khác nhau và chúng có khả năng tích hợp với nhau một cách chặt chẽ. Ngoài ra, với .NET Framework, các công ty còn có thể tận dụng những lợi thế của kỹ năng phát triển sẵn có mà không cần phải đào tạo lại và cho phép những người phát triển sử dụng ngôn ngữ mà họ ưa thích.

- ***Nâng cao năng suất cho các nhà phát triển*:** Với số lượng các nhà phát triển ứng dụng không nhiều nên mỗi giờ làm việc họ phải cho ra kết quả công việc cụ thể. Các nhóm phát triển với .NET Framework có thể loại bỏ những công việc lập trình không cần thiết và tập trung vào viết các lôgic doanh nghiệp. Chẳng hạn như .NET Framework có ưu điểm tiết kiệm thời gian như thực hiện các giao dịch tự động và dễ sử dụng, quản lý bộ nhớ một cách tự động và có chứa một tập các đối tượng điều khiển đa dạng bao hàm nhiều tác vụ phát triển chung.

- ***Bảo vệ những sự đầu tư thông qua việc bảo mật đã được cải tiến*:** Một trong những vấn đề liên quan lớn nhất đến Internet hiện nay là bảo mật. Kiến trúc bảo mật của .NET Framework được thiết kế từ dưới lên để đảm bảo các ứng dụng và dữ liệu được bảo vệ thông qua một mô hình bảo mật dựa-trên-bằng-chứng (evidence-based) và tinh vi.

- **T*ận dụng những dịch vụ của hệ điều hành*:** Windows cung cấp một số lượng đa dạng các dịch vụ có sẵn với bất kỳ nền tảng nào; như truy cập dữ

liệu một cách toàn diện, bảo mật tích hợp, các giao diện người dùng tương tác, mô hình đối tượng thành phần đáng tin cậy và các giám sát quá trình giao dịch. .NET Framework đã tận dụng lợi thế đa dạng và phong phú này để đưa ra cho mọi người theo cách dễ sử dụng nhất.

2.3 Các dịch vụ .NET :

Để thực thi mô hình .NET, một vài khối hợp nhất (building block) cơ sở phải được đặt đúng chỗ (các block này định rõ các dịch vụ Web được xây dựng như thế nào). Các dịch vụ này cố gắng để trợ giúp các nhà phát triển xây dựng các ứng dụng .NET. Microsoft định nghĩa các dịch vụ khối hợp nhất .NET sau đây:

Authentication: Khi sử dụng các công nghệ Authentication (chứng thực) cũng như Passport (hộ chiếu) của Microsoft các nhà phát triển tạo ra các dịch vụ cho riêng mình và bảo vệ các dịch vụ như mong muốn.

- **Messaging**: Các đặc tính Messaging (truyền thông điệp) của .NET được xây dựng trên MSN Hotmail Web ã dựa vào dịch e-mail, Microsoft Exchange Server 2000, và Instant Messaging (truyền thông điệp tức thì). Những hệ thống truyền thông điệp này và những đặc tính có thể được phân tán đến bất kì thiết bị nào do tính không phụ thuộc nền tảng của chúng.
- **Personalized Experience (kinh nghiệm cá nhân): .NET** cho người dùng nhiều kiểm soát hơn thông qua các qui tắc xử lý dữ liệu và quyền ưu tiên mà nó xác định rõ dữ liệu phải được di chuyển và quản lý như thế nào.
- **XML (Extensible Markup Language):** XML được xem như một ngôn ngữ chung mà nó cho phép dữ liệu được di chuyển từ dạng này sang dạng khác trong khi bảo trì tính toàn vẹn của nó. Cùng với SOAP, XML có thể cung cấp một dịch vụ linh hoạt để quản lý và điều khiển dữ liệu.

Một trong những thực thi (implementation) đầu tiên được đề nghị của dịch vụ .NET là Microsoft HailStorm. HailStorm là một dịch vụ trung tâm-người dùng (user-centric), nó cung cấp cho những người-dùng-cuối khả năng lưu trữ thông tin cá nhân như các cuộc hẹn, lịch hay các thông tin tài chính.

Kết quả việc một người dùng đăng kí dịch vụ này là thông tin được chia sẻ với các ứng dụng khác (thông tin dùng chung) theo ý muốn của họ, và nó trở thành một phần của người dùng khi điều khiển các hoạt động trên Web. Trong thời gian tới bạn sẽ được tiếp cận các dịch vụ tương tự được đề xuất trên Web bởi các công ty bán cho bạn thông qua việc đăng kí vào dịch vụ của họ. Như chúng ta đã thấy .NET đã chuẩn bị cho việc sử dụng một số công nghệ của Microsoft mà nó bắt đầu thích nghi và được sự chấp nhận bởi cộng đồng tin học và Internet.

2.4 Tác động của .NET đối với chuyên gia CNTT:

Chiến lược .NET có thể tác động đến các chuyên gia CNTT theo một số cách. Trước hết chúng ta hãy xem xét .NET có thể tác động như thế nào đến các nhà phát triển và sau đó đánh giá tác động của nó đối với những nhà quản trị hệ thống và các chuyên gia CNTT khác.

Những nhà phát triển cảm thấy có một tác động mạnh mẽ từ ý tưởng .NET. Để hiểu tác động này, đầu tiên chúng ta phải biết công việc phát triển ứng dụng đã thay đổi như thế nào. Trước đây các nhà phát triển xây dựng các ứng dụng trên các dịch vụ hệ thống cục bộ. Một ứng dụng riêng biệt được xây dựng để chạy trên các dịch vụ được cung cấp bởi một hệ điều hành riêng biệt. Trong hệ thống này những nhà phát triển đã có thể kiểm soát một cách cụ thể ứng dụng hoạt động như thế nào trên nền tảng đó. Những ứng dụng cho những nền tảng riêng biệt thì không liên lạc (truyền thông) tốt được với nhau. Giai đoạn thứ hai của sự thay đổi xuất hiện có nghĩa các nhà phát triển phải chuyển sang một mức độ khác, gọi là mức thứ n (n-tier). Điều đó cho phép các nhà phát triển tạo ra các ứng dụng mà nó hoạt động trên một mức mạng. Nói cách khác, sự phát triển xuất hiện liên tục từ các dịch vụ hệ thống cục bộ cho đến các dịch vụ mạng toàn cầu. Sự phát triển này đã tạo ra khả năng phát triển các phần mềm doanh nghiệp mà thực chất tập chung hơn vào công việc kinh doanh mà nó tạo ra năng suất làm việc cao hơn.

Chúng ta hiện đang ở vào giai đoạn tiếp theo của công cuộc thay đổi thông qua XML và SOAP, dịch vụ Web. Dịch vụ Web cho phép các ứng dụng tương tác với nhau thông qua Internet và cung cấp các dịch vụ tới người dùng hay các ứng dụng khác. Đây là một sự thay đổi cơ bản theo cách các ứng dụng đã được mô tả trước đây khi mà chúng ta nghĩ về những ứng dụng như một sản phẩm: bạn mua CD-ROM và cài đặt sản phẩm trên một máy tính. Một dịch vụ Web hoàn tất một vài kiểu giao dịch và trao đổi thông tin. Khi xây dựng trên XML, các dịch vụ Web có thể được sử dụng bởi bất kỳ người nào với bất kỳ thiết bị đơn lẻ nào tại bất kỳ thời điểm đã cho nào. Đặc tính này cho phép bất cứ số lượng tiến trình chuyên biệt (đặc trưng riêng biệt cho từng ứng dụng) nào xuất hiện liên tục (seamlessly) trên Internet không có sự can thiệp của người dùng.

Quy trình dịch vụ Web được hoàn thành bởi việc sử dụng cả hai đặc tính chương trình ghép nối lỏng và ghép nối chặt. Qui trình nghiệp vụ của việc tính toán n-tier (ghép nối chặt) được kết hợp với các chuẩn truyền thông điệp ghép nối lỏng và các phương pháp truy nhập dữ liệu trên Internet. Do ý tưởng .NET được tìm thấy trên cơ sở của XML và khái niệm dịch vụ Web, các nhà phát triển có một cách mới để tạo ra các ứng dụng mà nó hoạt động và tích hợp dễ dàng hơn. Sự thách thức của các nhà phát triển là tích hợp những khái niệm đó với cái mà nền tảng .NET được xây dựng trên nó.

Để cho các chuyên gia CNTT khác, ý tưởng .NET sẽ không thay đổi hoàn toàn công việc quản lý máy phục vụ CNTT. Các chuyên gia CNTT phải nhìn thấy sự giải toả chung trong việc quản trị bởi vì mô hình tính toán phân tán .NET đã được sẵn sàng, nhưng việc quản lý các máy phục vụ xí nghiệp (Enterprise server) .NET và các máy trạm sẽ hoạt động phần lớn theo cùng một cách.

2.5 Tác động của .NET đối với người dùng :

Nền tảng .NET có thể có tác động sâu sắc đến kinh nghiệm người dùng (theo hướng tích cực). Trước khi khảo sát khả năng này, chúng ta hãy xem xét mô hình tính toán hiện thời. Hiện tại, việc tính toán người dùng chủ yếu là nằm ở phần cứng và hệ điều hành. Những người dùng sở hữu những thiết bị phần cứng như PC, laptop hay PDA và họ cài đặt phần mềm và cấu hình các hệ thống đó. Dữ liệu chủ yếu được quản lý và thao tác (và cả dữ liệu đã mất) cũng trên các hệ thống đó. Do sự tăng trưởng của công nghệ, số lượng PC tại gia và văn phòng ngày càng tăng lên nhanh chóng. Có ai đã từng nghĩ rằng một người dùng có thể cần đến 13 Gbyte ổ cứng? Mô hình tính toán hiện thời gây ra nhiều vấn đề do số lượng người sử dụng máy tính của họ ngày càng nhiều. Người dùng phải chú ý đến dữ liệu và thiết bị của chính họ và Internet được coi không gì hơn là một thứ đồ chơi tô vẽ.

.NET có khả năng thay đổi cách tiếp cận. Do sức mạnh của Internet, người dùng không còn thấy cần thiết phải lưu giữ tất cả dữ liệu và phần mềm trên máy tính cá nhân của họ. Thay vào đó, dữ liệu và ngay cả việc sử dụng các ứng dụng có thể được lưu trữ trên các máy phục vụ trên Internet (thông thường chi phí ở đây là không đáng kể). Đặc tính này đã xoá bỏ trách nhiệm người dùng về mặt quản lý. Người dùng truy nhập và thao tác dữ liệu, nhưng người quản trị trên máy phục vụ quản lí công việc lưu giữ, sự chấp nhận lỗi và lập kế hoạch cấu hình. Người dùng không còn phải lưu giữ dữ liệu cục bộ. Một khi dữ liệu được đưa lên Internet, chiến lược .NET sẽ bắt đầu vận hành. Hãy nhớ rằng .NET cho ta một cách để truyền dữ liệu một cách thông suốt thông qua XML và SOAP.

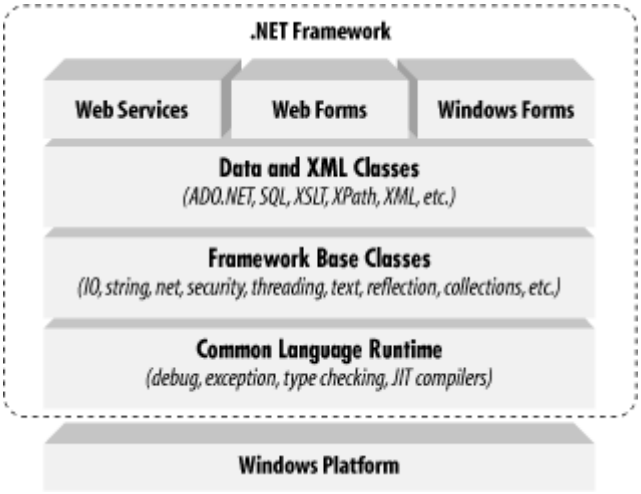
Hãy xem một ví dụ. Bạn đang làm việc trên một máy PC văn phòng và bạn quyết định mua một số cổ phiếu. Bạn truy nhập vào Website, sử dụng thông tin cá nhân của chính mình, có thể là với Microsoft HailStorm, và mua cổ phiếu. Vào một thời điểm muộn hơn, bạn đang đi trên đường trong giờ cao điểm. Bạn sử dụng cell-phone của bạn để kiểm tra giá cổ phiếu của bạn và

quyết định bây giờ là thời điểm để bán. Bạn sử dụng cell-phone để hoàn tất giao dịch. Như bạn có thể thấy, dữ liệu đang được thao tác trên chương trình máy phục vụ và bạn hoàn toàn không phải lo lắng gì về các trình ứng dụng hay tính tương thích nền tảng, và thông tin cá nhân của bạn đi theo bạn đến mọi nơi mọi chỗ mà bạn đến. Nhìn tổng thể, .NET làm cho Internet trở thành một nguồn dữ liệu tương tác thật sự, nơi mà người dùng quản lý tất cả các khía cạnh cuộc sống của họ từ công việc, vốn liếng cá nhân cho đến giải trí.

2.6 Kiến trúc .NET Framework :

.NET Framework được giới thiệu bao gồm 2 phần chính là Common Language Runtime (CLR) và Base Class Libraries (BCL). Một trong các thành phần này đều có vai trò cực kỳ quan trọng trong việc phát triển các dịch vụ và các ứng dụng .NET.

Common Language Runtime (gọi tắt là bộ thực thi) được xây dựng trên các dịch vụ hệ điều hành. Nó chịu trách nhiệm thực hiện các ứng dụng và đảm bảo các phần liên quan đến ứng dụng đều được đáp ứng như quản lí bộ nhớ, an ninh bảo mật, tích hợp ngôn ngữ..v.v. Bộ thực thi bao gồm nhiều dịch vụ hỗ trợ phát triển và triển khai ứng dụng cũng như cải thiện tính đáng tin cậy của ứng dụng. Tuy vậy, những nhà phát triển trên thực tế không tương tác với CLR. Họ sử dụng một tập các thư viện lớp cơ sở được xây dựng bên trên bộ thực thi này thông qua các ngôn ngữ lập trình. Như là một phần của các lớp cơ sở, .NET Framework còn cung cấp một mô hình lập trình ứng dụng Web ASP.NET và Windows Forms (để xây dựng các ứng dụng trên Windows). Riêng ASP.NET cung cấp các thành phần và các dịch vụ ở mức cao hơn nhằm mục đích phát triển các dịch vụ và các ứng dụng Web XML.



Hình 2.1 : Sơ đồ kiến trúc .NET Framework

2.6.1 Common Language Runtime :

Như các bạn đã biết, các ngôn ngữ lập trình khác nhau đều cung cấp một runtime (bộ thực thi) và .NET Framework không phải là một ngoại lệ. Tuy vậy bạn sẽ thấy bộ runtime này là khá đặc biệt so với phần lớn các runtime chúng ta vẫn sử dụng. Common Language Runtime trong .NET Framework quản lý sự thực hiện mã và cung cấp sự truy cập vào nhiều loại dịch vụ giúp cho quá trình phát triển được dễ dàng hơn. CLR đã được phát triển ở tầm cao hơn so với các runtime trước đây như VB-runtime chẳng hạn, bởi nó đạt được những khả năng như tích hợp các ngôn ngữ, bảo mật truy cập mã, quản lý thời gian sống của đối tượng và hỗ trợ gỡ lỗi

Mã được biên dịch và hướng tới CLR có tên "managed code". "Managed code" cung cấp siêu dữ liệu (metadata) cần thiết cho CLR để cung cấp các dịch vụ hỗ trợ đa ngôn ngữ, bảo mật mã, quản lý thời gian sống của đối tượng và quản lý bộ nhớ.

Trình biên dịch và quản lý mã

.NET Framework đòi hỏi bạn phải sử dụng một trình biên dịch ngôn ngữ "nhúng" vào CLR như trình biên dịch Visual Basic .NET, C# .NET, C++ .NET hay JScript .NET của Microsoft. Chú ý rằng có nhiều trình biên dịch của đối tác thứ ba có trên thị trường như COBOL và Perl.

Sau khi sử dụng một trong các trình biên dịch ngôn ngữ, mã của bạn được biên dịch sang Microsoft Intermediate Language (MSIL hay đơn giản IL). IL là một tập các lệnh độc lập CPU có thể được chuyển đổi dễ dàng sang "native code". Siêu dữ liệu cũng được chứa bên trong IL.

IL độc lập với CPU có nghĩa mã IL không đáng tin cậy trên máy tính đặc thù sinh ra nó. Nói cách khác, nó có thể được chuyển từ một máy này sang một máy khác (miễn là máy tính hỗ trợ .NET Framework) mà không gặp bất cứ khó khăn nào. Sau IL, mã mà bạn bắt đầu với nó sẽ được biên dịch bởi trình biên dịch JIT (Just-In-Time) tới mã máy hay "native code". IL chứa đựng mọi thứ cần thiết để làm điều này như nạp các chỉ lệnh và gọi các hàm (call methods) và một số các thao tác khác.

Trình biên dịch JIT

.NET Framework chứa đựng một hay nhiều trình biên dịch JIT có thể biên dịch mã IL của bạn ra mã máy hay mã cho CPU cụ thể. Điều đó được thực hiện khi ứng dụng chạy lần đầu tiên.

Bạn hãy chú ý quá trình này khi bạn xây dựng trang ASP.NET đầu tiên của mình. Sau khi bạn xây dựng một trang ASP.NET bất kỳ, bạn biên dịch trang này sang IL. Khi sử dụng trình duyệt và gọi trang bằng cách gõ URL của nó vào thanh địa chỉ, bạn chú ý một sự tạm dừng không đáng kể khoảng vài giây. Trên thực tế nó đang gọi mã IL và chuyển nó với một trình biên dịch JIT để để biên dịch sang mã máy. Điều đó chỉ xảy ra lần đầu khi một ai đó yêu cầu trang này. Sau lần đầu, bạn có thể gõ F5 để "refresh" lại trạng này và trang được thực hiện ngay lập tức. Trang đã được chuyển đổi sang mã máy và bây giờ được lưu giữ trong bộ nhớ. CLR biết rằng trình biên dịch JIT đã biên dịch trang. Chính vì thế, nó nhận được đầu ra của trang từ bộ nhớ. Nếu về sau bạn làm một sự thay đổi trang ASP.NET của bạn, hãy biên dịch lại, sau đó chạy trang lần nữa, CLR phát hiện ra có sự thay đổi file gốc. Nó sử dụng trình biên dịch JIT một lần nữa để biên dịch mã IL sang mã máy.

Trình biên dịch JIT đảm bảo, trong quá trình biên dịch sang mã máy, mã được an toàn kiểu (type safe). Điều đó có nghĩa những đối tượng luôn tách rời và chúng không cố ý làm hỏng một đối tượng khác.

Các Assembly

Trong các ứng dụng bạn xây dựng bên trong .NET Framework, các assembly luôn đóng một vai trò quan trọng. Các assembly có thể được hiểu như các khối hợp nhất (building block) của các ứng dụng của bạn. Nếu thiếu một assembly liên quan, mã sẽ không thể biên dịch được từ IL. Khi bạn đang sử dụng trình biên dịch để biên dịch mã từ mã được quản lý (managed code) sang mã máy, trình biên dịch JIT sẽ tìm kiếm mã IL được cất giữ trong một file PE cùng "assembly manifest" (bản kê khai assembly) có liên quan. Cứ mỗi lần bạn tạo một ứng dụng Web Form hay Windows Form trong .NET, thực tế bạn đang tạo ra một assembly. Cứ mỗi ứng dụng trong các ứng dụng này có chứa tối thiểu một assembly.

Trong Windows DNA, nơi các DLL và EXE là những "khối hợp nhất" (building block) của các ứng dụng, trong .NET, nó là assembly được sử dụng như một cơ sở cho các ứng dụng.

Trong Windows DNA và COM, có những trường hợp được tham chiếu đến như DLL hell. Các thành phần COM nói chung được thiết kế để chỉ có một phiên bản của thành phần COM này trên một máy tính tại bất kỳ thời gian đã cho nào. Bởi vì đặc tả COM không bao gồm cả thông tin phụ thuộc trong một định nghĩa kiểu của thành phần. Với .NET, điều đó bây giờ có thể có nhiều phiên bản của các component, hay các assembly, chạy trên cùng một server cạnh nhau. Một ứng dụng sẽ luôn luôn tìm kiếm assembly xây dựng nó.

Khi một ứng dụng được bắt đầu trong .NET, ứng dụng sẽ tìm kiếm một assembly trong thư mục cài đặt. Các assembly được lưu giữ trong một thư mục cài đặt được trỏ đến đến các private assembly. Nếu ứng dụng không tìm ra assembly trong thư mục cài đặt, ứng dụng sẽ quay ra GAC (Global Assembly Cache) để tìm chúng.

Cấu trúc của một assembly

Các assembly chứa mã được thực hiện bởi Common Language Runtime. Cái được lớn nhất của assembly là chúng "tự mô tả" (self-describing). Tất cả những chi tiết về assembly được cất giữ bên trong bản thân assembly đó. Trong Windows DNA, COM cất giữ tất cả dữ liệu "tự mô tả" của nó trong nơi đang ký (registry) của server, và như vậy việc cài đặt cũng như loại bỏ các thành phần COM có nghĩa dừng hoạt động (shutting down) IIS. Bởi vì một assembly .NET cất giữ thông tin này bên trong bản thân nó, nó có thể thực hiện chức năng XCOPY. Cài đặt một assembly cũng đơn giản như copy nó và điều đó là không cần thiết để dừng hay bắt đầu IIS trong nó đang hoạt động.

Các assembly được tạo ra bởi các phần sau :

- Bản kê khai assembly (Assembly manifest)
- Kiểu siêu dữ liệu (Type metadata)
- Mã MSIL (Microsoft Intermediate Language code)

Assembly manifest là nơi các chi tiết của assembly được cất giữ. Còn assembly được cất giữ bên trong bản thân DLL hay EXE. Các assembly có thể là những file assembly đơn hay nhiều file, và bởi vậy assembly các assembly manifest được cất giữ trong assembly hay trong một file rời. assembly manifest cũng cất giữ số phiên bản của assembly để đảm bảo rằng ứng dụng luôn luôn được sử dụng đúng phiên bản. Khi bạn có nhiều phiên bản của một assembly trên cùng một máy tính, điều đó là rất quan trọng để gán nhãn chúng thật cẩn thận, như vậy CLR biết được phiên bản nào đang được sử dụng.

Các số phiên bản trong các assembly được xây dựng theo các cách sau: <major version>.<minor version>.<build number>.<revision>

Kiểu siêu dữ liệu (Type metadata) đã được hiểu như "dữ liệu về dữ liệu" (data about data). Siêu dữ liệu này chứa thông tin trên những kiểu được

đưa ra bởi assembly như thông tin cho phép bảo mật, thông tin về giao diện và lớp và các thông tin về assembly khác.

Garbage collection (gom rác)

.NET Framework là một môi trường "gom rác". "Gom rác" là quá trình của sự phát hiện khi các đối tượng không còn được sử dụng và tự động phá hủy các đối tượng này, như vậy bộ nhớ được giải phóng.

"Gom rác" không phải là một khái niệm mới. Nó đã được sử dụng trong các ngôn ngữ khác từ khá lâu rồi. Trong thực tế, Java đã có một hệ thống "gom rác" đúng chỗ. Các ngôn ngữ khác như C++ không có "gom rác". Những nhà phát triển C++ được yêu cầu rất cẩn thận khi hủy bỏ một đối tượng và giải phóng bộ nhớ. Điều đó dẫn đến một số vấn đề thiếu hụt bộ nhớ. Nếu nhà phát triển quên giải phóng các đối tượng từ ứng dụng, việc cấp phát bộ nhớ của ứng dụng ngày càng tăng, đôi khi là rất đáng kể. Việc giải phóng các đối tượng quá sớm sẽ nảy sinh lỗi ứng dụng; các kiểu lỗi này, trong phần lớn trường hợp, thường rất khó để theo dõi.

Trong .NET, bộ "gom rác" cho phép bạn khi làm việc mà không phải theo dõi mã cho những đối tượng không cần thiết và phá hủy chúng. bộ "gom rác" sẽ chú ý tất cả những điều này cho bạn. "Gom rác" không xảy ra ngay lập tức, nhưng thay vào đó bộ "gom rác" thỉnh thoảng sẽ "vun đống" để xác định những đối tượng nào sẽ phải phá hủy. Hệ thống mới này hoàn toàn miễn trách nhiệm cho người phát triển luôn phải tìm cách để sử dụng cũng như giải phóng bộ nhớ.

Với bộ "gom rác" mới này, bạn có thể điều khiển những khía cạnh nhất định của những hàm của nó, như nó làm việc "sau hậu trường" trong ứng dụng của bạn. Trong SDK documentation, tìm lớp System.GC để có thêm thông tin.

2.6.2 Base Class Libraries :

Thư viện các lớp cơ sở .NET Framework cung cấp một tập các lớp ("APIs"), hướng đối tượng, có thứ bậc và có thể mở rộng và chúng được sử

dụng bởi bất cứ ngôn ngữ lập trình nào. Như vậy, tất cả các ngôn ngữ từ Jscript cho tới C++ trở nên bình đẳng, và các nhà phát triển có thể tự do lựa chọn ngôn ngữ mà họ vẫn quen dùng.

Tập các lớp, các kiểu giá trị và giao diện này được tổ chức bằng một hệ thống các Namespace (xem định nghĩa phần dưới). Bảng 1 dưới đây đưa ra một mô tả chi tiết những Namespace có sẵn trong .NET Framework. Một điều rất quan trọng là chúng ta không chỉ giới hạn ở các Namespace này. Bạn có thể tự tạo ra Namespace và sử dụng chúng trong ứng dụng của bạn hay cũng có thể sử dụng các Namespace của đối tác thứ ba đang có đầy trên thị trường. Một ví dụ cho trường hợp này là Namespace System.Data.Oracle.

2.6.3 ASP.NET :

ASP.NET là một tập các lớp nằm trong thư viện lớp cơ sở. ASP.NET cung cấp một mô hình ứng dụng Web dưới dạng một tập các control (đối tượng điều khiển) và cơ sở hạ tầng giúp bạn tạo ra các ứng dụng Web một cách dễ dàng. Các control này được xây dựng cho các ứng dụng trên máy phục vụ (hay còn gọi là Web Forms) phản ánh những control giao diện người dùng HTML đặc thù như Listbox, Textbox và Button và một tập bổ sung các control Web phức tạp hơn như calendars chẳng hạn. Một đặc tính quan trọng của các control trên là chúng được viết để thích nghi với những khả năng của các ứng dụng máy khách. Nói cách khác các đối tượng điều khiển Web forms có thể "đánh hơi" thấy máy khách đang yêu cầu một trang (page) và trả lại người dùng một cách thích hợp (ví dụ như WML cho phone hay Dynamic HTML cho Internet Explorer 5.5).

ASP.NET cũng cung cấp những đặc tính như quản lý trạng thái "cluster session" và phục hồi tiến trình giúp giảm bớt số lượng mã mà một người phát triển phải viết và tăng độ tin cậy ứng dụng. Ngoài ra ASP.NET cho phép các nhà phát triển chuyển giao phần mềm như là một dịch vụ. Khi sử dụng những đặc tính các dịch vụ Web XML ASP.NET, các nhà phát triển ASP.NET có thể viết những giao dịch logic đơn giản và cơ sở hạ tầng ASP.NET sẽ chịu

trách nhiệm chuyển dịch vụ đó theo đường SOAP và các giao thức công cộng khác. ASP.NET làm việc với mọi ngôn ngữ và công cụ (bao gồm cả Visual Basic, C++, C#, và JScript).

Bên trong ASP.NET

Tại lõi của ASP.NET là HTTP-runtime (bộ thực thi HTTP), một "động cơ" thực hiện với hiệu suất cao khi xử lý các lệnh HTTP. HTTP-runtime có trách nhiệm xử lý tất cả những yêu cầu HTTP gửi đến, giải quyết URL của mỗi yêu cầu tới một ứng dụng, và sau đó gửi yêu cầu tới ứng dụng cho việc xử lý sau này. HTTP-runtime có tính đa luồng và xử lý các yêu cầu không đồng bộ. Hơn nữa, HTTP-runtime được thiết kế mang tính phục hồi cao, như vậy nó có thể phục hồi một cách tự động từ những vi pham khi truy nhập.

Ngoài ra ASP.NET còn có khả năng như cập nhật ứng dụng, có thể mở rộng, quản lý và cất giữ trạng thái và nhiều tính năng cao cấp khác.

CHƯƠNG 3: .NET FRAMEWORK TRONG BẢO MẬT

3.1 .NET Framework và Common Language Runtime :

.NET Framework và Common Language Runtime (CLR) cho phép người lập trình làm việc đơn giản hơn với các phương pháp bảo mật.Giả sử như chống lại việc thông tin bị đánh cắp bằng cách sử dụng phương pháp bảo mật thích hợp trong chương trình.Các mã độc có thể ngăn chặn bằng việc phân quyền người sử dụng và bảo vệ truy cập mã nguồn Code Access Security (CSA).Kiểu tấn công làm tràn bộ đệm trở nên không tưởng thông qua việc thực hiện các vấn đề về an ninh và quản lý chặt chẽ môi trường chạy ứng dụng (runtime enviroment) được cung cấp bởi nền tảng .NET. Các lỗi trong kiểu tấn công tràn bộ đệm đã được loại bỏ bằng việc quản lý mã và CLR kiểm tra môi trường chạy ứng dụng

Dưới đây là một số các tính năng được cung cấp bởi nền tảng .NET khá quan trọng trong vấn đề an ninh và mã hóa:

- Thiết lập các chính sách an ninh và tính xác thực (Kiểm soát qua an ninh của .NET)
- CSA (Quyền thi hành dựa trên tính xác thực và chính sách an ninh)
- Role-based security (Quyền truy cập thông qua việc xác thực người dùng và vai trò của người dùng)
- Quản lý và xác thực môi trường ứng dụng (Kiểm tra dải địa chỉ và kiểu)
- Lớp Cryptography (Cung cấp các thuật toán mã hóa an toàn)

3.2 .NET Framework đơn giản hóa việc bảo mật như thế nào :

Một vấn đề lớn trong lập trình các phương pháp an ninh sử dụng Win32 API là rất khó khăn trong để hiểu và sử dụng.Những dòng mã khó hiểu được sắp xếp lại để xử lý một cách đơn giản nhất, ví dụ như việc lấy khóa trong Cryptographic service provider (CSP) mà rất nhiều người phát triển thường bỏ qua nó trong khi họ có thể tiến xa cùng với nó. Những phát triển thường áp

dụng chính sách an ninh bằng việc dùng Win32 API, họ thường phải cố gắng làm với một lối lập trình phức tạp.

.NET Framework cung cấp rất nhiều phương pháp đơn giản bằng cách đưa Win32 Security API vào một đối tượng đơn giản hơn. Rất nhiều phép toán như lấy khóa từ trong CSP giờ có thể tự động lấy khóa trong lớp an ninh của .NET Security Framework. Thêm nữa mỗi lớp trong .NET Security Framework được tích hợp các tính năng an ninh được và khai báo như lớp được niêm phong cho nên không thể đánh cắp và bị lộ.

3.3 Độ tin cậy và nền tảng .NET :

Trước khi bạn quyết định sử dụng bất ký một biện pháp an ninh hay các kỹ thuật mã hóa, bạn phải chắc chắn về độ tin cậy của chương trình. Nền tảng .NET dã tiến một bước xa để giải quyết vấn đề vệ độ tin cậy. Đầu tiên, rất quan trọng để chấp nhận rằng ứng dụng .NET không được biên dịch thành mã có thể nhìn thấy được. Mặc dù nó được biên dịch thành một loại mã trung gian được hiểu như mã trung gian của Microsoft, rất giống như loại mã sử dụng trong nền tảng Java. Nó cho phép CLR và .NET Framework xử lý rát nhiều dịch vụ an ninh tự động đáng tin cậy như:

- Kiểm tra giới hạn trong quá trình chạy chương trình để tránh sự thất thoát bộ nhớ và tràn stack
- Kiểm tra kiểu dữ liệu trong quá trình chạy chương trình để tránh đưa ra kiểu dữ liệu sai
- Đi dọc stack để kiểm tra sự cho phép để gọi mã
- Tự động thu gom rác một cách hiệu quả để tiết kiệm bộ nhớ
- Kiểm soát lỗi tránh những lỗi bất thường trong quá trình chạy
- Bảo vệ theo vai trò để xác thực và giới hạn thực thi cho người sử dụng
- Bảo vệ theo chứng thực để quản lý việc cho phép sử dụng mã cơ sở

3.4 Quản lý mã và định kiểu an toàn :

Mã có thể sử dụng dịch vụ của CLR có tên là quản lý mã. CLR cung cấp một tập hợp dịch vụ, như kiểm tra định kiểu an toan và tự động thu gom rác, làm tăng tính tin cậy và tính an toàn. Để sử dụng được những dịch vụ của CLR, quản lý mã cần phải dự đoán trước, sắp xếp, và kiểu đồng nhất. Định kiểu an toàn là một khía cạnh quan trọng trong tính an toàn và bảo mật. Định kiểu an toàn hoàn toàn có thể thực hiện được vì CLR hiểu chi tiết về loại dữ liệu nào đang được quản lý. Sử dụng sự hiểu biết đó, CLR có thể ép kiểu chính xác để đưa ra các luật trong việc định kiểu an toàn.

Ví dụ, tất cả các loại dữ liệu bao gồm cả chuỗi và mảng có một kiểu giống nhau và đồng nhất. Common Type System (CTS) định nghĩa các luật cho mỗi kiểu dữ liệu được quản lý, nó tốt giống như quá trình xử lý của CLR định kiểu cho các kiểu dữ liệu đó. Sự giới hạn của các luật được đưa ra bởi CTS và được thực thi bởi MSIL. CTS cũng định nghĩa mỗi loại dữ liệu đưa ra và quá trình xử lý chấp nhận trong việc quản lý mã. CTS đưa ra giới hạn các lớp trong việc xử lý các lớp kế thừa và chống lại việc xử lý dữ liệu không an toàn, như việc đưa kiểu số nguyên vào con trỏ và tràn giới hạn của mảng. Mã MISL là trình biên dịch quá trình chạy thành một bảng hướng dẫn chi tiết về thiết bị phần cứng trước khi kiểu dữ liệu được kiểm tra.

Để thực hiện việc kiểm tra kiểu an toàn, .NET tích hợp thêm các metadata để định nghĩa mã nguồn và dữ liệu chứa trong chương trình. Trình quản lý mã được tự động chấp nhận hoặc từ chối bởi CLR. Trình quản lý bộ nhớ tự động này gần giống với “việc thu gom rác”. Thu gom rác chống lại việc rò rỉ bộ nhớ và tăng độ tin cậy

Tất cả các đối tượng đều có kiểu, và tùy thuộc vào mỗi tham chiếu đến đối tượng để định nghĩa việc sắp xếp bộ nhớ. Từ khi việc tùy ý xử lý con trỏ bị bỏ (trừ trường hợp từ khóa không an toàn được dùng), thì chỉ có một cách duy nhất để truy cập vào đối tượng thông qua các thành phần công cộng. Tuy nhiên, CLR có thể xác nhận sự an toàn của một đối tượng bằng cách phân tích

mỗi metadata. Vì vậy không cần phải phân tích toàn bộ mã nguồn mà chỉ cần phân tích đối tượng là có thể kết luận tính an toàn.

Chúng ta cũng có thể không sử dụng trình quản lý mã trong ngôn ngữ C# bằng cách sử dụng từ khóa *unsafe*, nhưng một số các ngôn ngữ khác như VB.NET chỉ sử dụng được khi chúng ta sử dụng định kiểu an toàn và trình quản lý mã. Từ khóa *unsafe* chỉ cần thiết khi chúng ta làm việc trực tiếp với con trỏ bộ nhớ. Mã không được quản lý chỉ hữu dụng khi chúng ta gọi hàm kế thừa DLLs, sử dụng PInvoke nhuần nhuyễn, nhưng Mã không được quản lý sẽ không được thẩm định bởi định kiểu an toàn của CLR.

MSIL định nghĩa nền thực thi sử dụng trong tất cả các trình biên địch .NET. Ngôn ngữ này thực thi và đòi hỏi các kiểu được định nghĩa bởi CTS trong quá trình chạy. MSIL thì chuyển đổi một nền nào đó thành mã đơn giản sau đó CLR sẽ thực hiện việc kiểm tra kiểu. Kiểm tra kiểu được tiến hành một cách mặc định, nhưng nó có thể bỏ qua nếu mã đó đáng tin cậy.

CHƯƠNG 4: LỚP CRYPTOGRAPHY VÀ MỘT SỐ LOẠI MÃ HOÁ ĐƯỢC HỖ TRỢ BỞI .NET

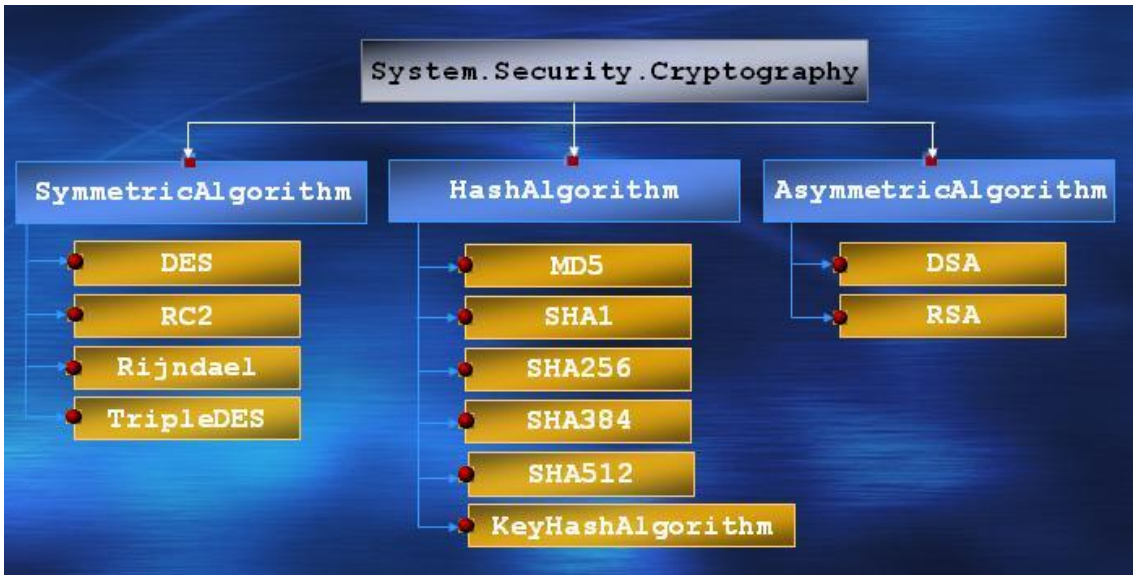
4.1 Tổng quan lớp Cryptography :

Microsoft đăng ký quyền sở hữu Win 32 Cryptography API (CryptoAPI) vào năm 1996 tích hợp trong Win NT. Mặc dù CryptoAPI cung cấp đầy đủ sự hỗ trợ cho nghành lập trình mật mã, nhưng nó rất khó sử dụng. Bạn phải hiểu thật rõ về mật mã học và phải làm việc với rất nhiều các thành phần cũng như những con số lớn trong API. Nó cũng không phải là một đối tượng được định hướng cho đến khi nó xuất hiện trong ngôn ngữ lập trình C, và bạn có thể gọi hàm để sử dụng nó một cách đơn giản. Thật may mắn, .NET Framework làm đơn giản hóa việc sử dụng API của chúng ta bằng cách cung cấp một lớp rất thân thiện với tên gọi **System.Security.Cryptography**

Chúng ta sẽ cùng tìm hiểu ngắn gọn một số một số lớp chính có trong lớp **System.Security.Cryptography**

- **SymmetricAlgorithm :** Lớp mã hóa đối xứng. Nó sẽ mã hóa theo các thuật toán mã hóa đối xứng như: DES, Rịndael.
- **AsymmetricAlgorithm :** Lớp mã hóa bất đối xứng, nó sẽ mã hóa theo các thuạt toán RSA, DSA.
- **CryptoStream :** Kết nối dòng dữ liệu nguồn với các thuật toán mã hóa.
- **CspParameters :** Chứa thông tin của các tham số trong những thuật toán đặc biệt để có thể lưu trữ và lấy lại thông qua *Cryptographic Service Provider* (CSP).
- **HashAlgorithm :** Lớp cơ sở hỗ trợ các thuật toán băm.
- **RandomNumberGenerator :** Lớp cơ sở sinh ra số ngẫu nhiên.
- **ToBase64Transform** và **FromBase64Transform :** Dùng đẻ chuyển đổi các dãy Byte và Base-64.
- **CryptographicException :** Chứa thông tin về lỗi của các loại mã hóa khác nhau.

Các bạn có thể nhớ và tùy ý sử dụng các lớp này trong chương trình của mình vì nó đã có sẵn trong lớp **System.Security.Cryptography**



Hình 4.1 : Các thuật toán mã hóa hỗ trợ trong lớp Cryptoghraphy

4.2 Các thuật toán mã hóa đối xứng trong .NET :

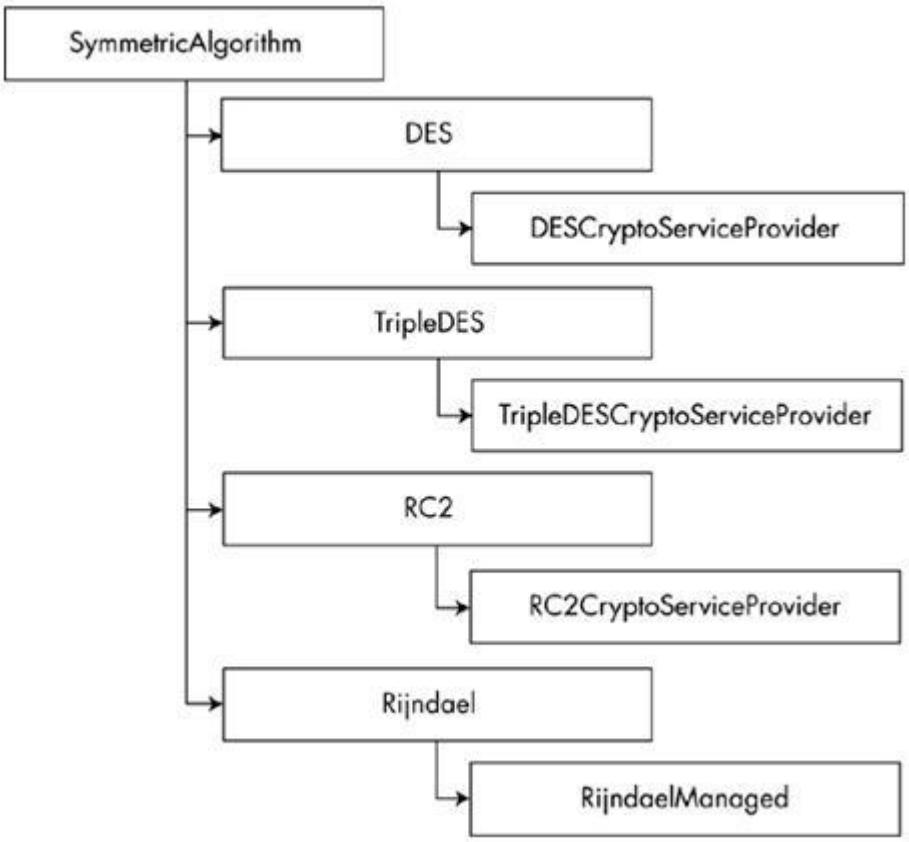
Các lớp .NET Framework thực thi thuật toán mã hóa đối xứng thông qua lớp cơ sở **SymmetricAlgorithm**. Lớp **SymmetricAlgorithm** có một vài trường được khai báo "protected" nên không thể truy vấn trực tiếp vào phương thức không được cung cấp bởi lớp. Tuy nhiên những trường này có thể truy vấn thông qua các thuộc tính ảo, thuộc tính ảo thực thi cụ thể trên lớp cụ thể. Ví dụ như trường số nguyên **BlockSizeValue** có thể truy vấn thông qua thuộc tính ảo **BlockSize** theo kiểu thích hợp, dựa trên lớp thực tế đang sử dụng. Theo cách này, việc cố gắng đặt cỡ của khối thành giá trị cụ thể là không hợp pháp trong thuật toán mã hóa đối xứng, sẽ được cho phép bởi **CryptographicException**, dựa trên thuật toán cụ thể đang được sử dụng. Trong mỗi trường hợp trường được khai báo "protected" và thuộc tính ảo sẽ có chung kiểu dữ liệu, và tên sẽ cùng được loại bỏ, chỉ có giá trị gắn liền với trường được khai báo "protected" được giữ lại.Các thuộc tính ảo có trong lớp **SymmetricAlgorithm** thể hiện trong bảng sau:

Thuộc tính ảo	Ý nghĩa
BlockSize	Lấy hoặc đặt giá trị cho kích cỡ khối dưới dạng các bit cho thuật toán, để dữ liệu tổng có thể được mã hóa hoặc giải mã trong 1 bước. Dữ liệu mà lớn hơn kích cỡ khối sẽ được chia ra các khối khác có kích cỡ giống như khối đã tạo. Khối cuối cùng sẽ được gắn thêm kích cỡ của nó. Các cỡ khối thích hợp sẽ được phân biệt bởi **LegalBlockSizes** trong mỗi thuật toán.Kiểu **int**
FeedbackSize	Lấy hoặc đặt kích cỡ của giá trị trả về dưới dạng các bit cho mỗi thuật toán, nơi mà kết quả cuối cùng được trả về sau khi đã mã hóa hoặc giải mã. Nó là bắt buộc trong phương thức OFB và CFB của quá trình tính toán. Kích cỡ thích hợp sẽ dựa trên các thuật toán mã hóa đối xứng, nhưng nó không được lớn hơn kích cỡ của khối. Và nó có kiểu **int**
IV	Lấy hoặc đặt giá trị ban đầu cho vector trong thuật toán mã hóa đối xứng, bắt buôc phải chó trong phương thức CBC. Có kiểu mảng Byte
Key	Lấy hoặc đặt giá trị cho khóa bí mật sử dụng trong thuật toán mã hóa đối xứng để mã hóa hoặc giải mã. Có kiểu **mảng Byte**
KeySize	Lấy hoặc đặt cỡ cho khóa bí mật dưới dạng các bit. Kích cỡ thích hợp sẽ được phân biệt bởi **LegalKeySizes** trong mỗi thuật toán. Có kiểu **int**
LegalBlockSizes	Lấy kích cỡ của khối được hỗ trợ bởi các thuật toán mã hóa đối xứng
LegalKeySizes	Lấy kích cỡ của khóa được hỗ trợ trong thuật toán mã hóa đối xứng
Mode	Lấy hoặc đặt chế độ để thực hiện trong các thuật toán. Nó có kiểu là kiểu **CipherMode**.VD: ECB, CBC, CFB, OFB...
Padding	Lấy hoặc dặt giá trị chèn vào các byte còn trống của khối cuối cùng. Có kiểu **PaddingMode**. VD: PKCS7, Zeos, None

SymmetricAlgorithm được thiết kế là lớp "public" và không chứa tham số. Kiểu thiết kế này tạo ra các khóa bí mật khác nhau. Tất nhiên, **SymmetricAlgorithm** cũng hỗ trợ các phương thức chuẩn **Equals, Finalize, GetHashCode, ToString, GetType**, và **MemberwiseClone,** những phương thức mà đã được định nghĩa ở lớp **Object** cơ sở.

Phương thức chung	Ý nghĩa
Clear	Sẽ gọi **Dispose**, giải phóng nguồn được sử dụng trong thuật toán mã hóa đối xứng. Phương thức trả về kiểu **void**
Create	Tạo đối tượng **SymmetricAlgorithm** để mã hóa hoặc giải mã. Phương thức trả về đối tượng **SymmetricAlgorithm**
CreateDecryptor	Tạo đối tượng giải mã sử dụng khóa và vector khởi tạo. Phương thức tham chiếu đến **ICryptoTranform** sử dụng để chuyển dữ liệu thành các khối
CreateEncryptor	Tạo đối tượng mã hóa sử dụng khóa và vector khởi tạo. Phương thức tham chiếu đến **ICryptoTranform** sử dụng để chuyển dữ liệu thành các khối
Equals	Kế thừa từ lớp **Object**, sử dụng để so sánh 2 đối tượng **SymmetricAlgorithm** cho bằng nhau. Giá trị trả về có dạng **bool**
GenerateIV	Khởi tạo vector bất kỳ. Trả về kiểu **void**
GenerateKey	Khởi tạo khóa bất kỳ. Trả về kiểu **void**
GetHashCode	Kế thừa từ lớp **Object**, cung cấp giá trị băm cho đối tượng **SymmetricAlgorithm**. Trả về kiểu **int**
GetType	Kế thừa từ lớp **Object** sử dụng để lấy kiểu cho đối tượng **SymmetricAlgorithm**. Trả về kiểu **Type**
ToString	Kế thừa từ lớp **Object**, sử dụng để cung cấp chuỗi hiển thị cho đối tượng **SymmetricAlgorithm**
ValidKeySize	Phương thức này quyết định khi kích cỡ khóa phù hợp với thuật toán đang sử dụng. Trả về kiể **bool**

Bạn sẽ không làm việc trực tiếp với đối tượng **SymmetricAlgorithm**, vì nó là một đối tượng trìu tượng. Bạn sẽ làm việc với các class được cung cấp và nó hoạt động như một phương thức ảo của **SymmetricAlgorithm** dưới đây là sơ đồ các lớp trong **SymmetricAlgorithm**



Hình 4.2 : Thuật toán mã hóa đối xứng trong lớp SymmetricAlgorithm

Chúng ta có thể thấy trong Hình 4.2 là các lớp có trong lớp **Symmetric Algorithm**, chúng cũng là các lớp trìu tượng. Bây giờ chúng ta sẽ cùng tìm hiểu ý nghĩa của các lớp.

- **DES** là lớp trìu tượng đóng gói theo thuật toán mã hóa đối xứng DES
- **TripleDES** là lớp trìu tượng đóng gói theo thuật toán mã hóa đối xứng Triple DES, thuật toán này có độ an toàn cao hơn DES
- **Rijndael** là lớp trìu tượng đóng gói theo thuật toán mã hóa đối xứng Rijndael nó là một chuẩn mới thay thế DES

- **RC2** là lớp trìu tượng đóng gói theo thuật toán mã hóa đối xứng RC2 được Ronald Rivest nghiên cứu để thay thế DES

Thuật toán	Kích thước khóa hợp lệ	Kích thước khóa mặc định
DES	64 bits	64 bits = 8 bytes
RC2	Từ 40 đến 128 bits	128 bits = 16 bytes
Triple DES	128, 192 bits	192 bits = 24 bytes
Rijndael	128, 192, 256 bits	256 bits = 32 bytes

Chúng ta cùng xét 1 ví dụ về cách sử dụng thuật toán mã hóa đối xứng trong .NET

-Sinh khóa bí mật :

```
static string GenerateKey()
{
    DESCryptoServiceProvider myDES ;
    myDES = new DESCryptoServiceProvider();
    //myDES.GenerateKey();

    return ASCIIEncoding.ASCII.GetString(myDES.Key);
}
```

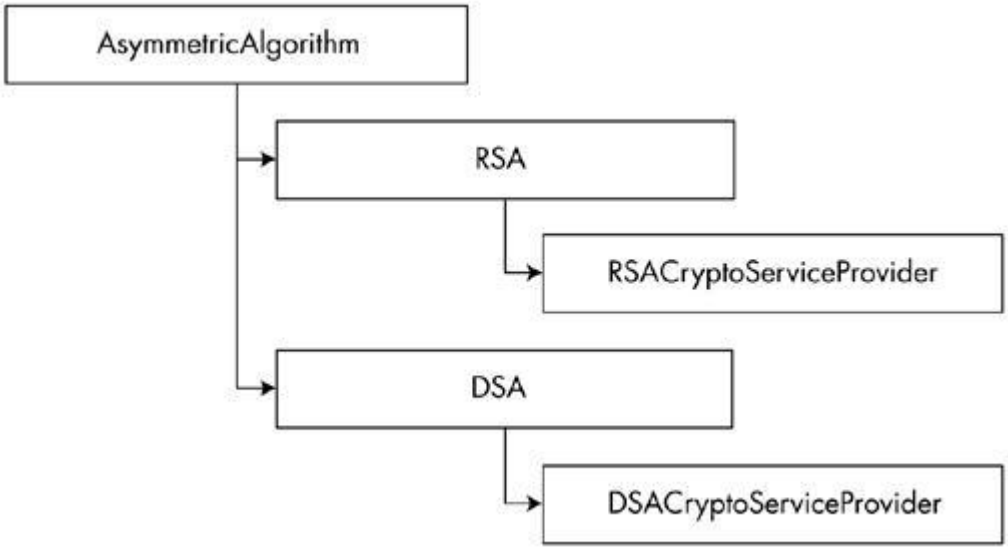
-Mã hóa file :

```
void EcryptFile(string inputFile, string outputFile, string szSecureKey)
{
    FileStream inStream, outStream;
    inStream = new FileStream(inputFile, FileMode.Open,
FileAccess.Read);
    outStream = new FileStream(outputFile,FileMode.Create,
FileAccess.Write);
```

```
        DESCryptoServiceProvider myDES = new
DESCryptoServiceProvider();
        myDES.Key = ASCIIEncoding.ASCII.GetBytes(szSecureKey);
        myDES.IV = ASCIIEncoding.ASCII.GetBytes(szSecureKey);
        ICryptoTransform myDES_Ecryptor = myDES.CreateEncryptor();
        CryptoStream myEncryptStream;
        myEncryptStream = new CryptoStream(outStream, myDES_Ecryptor,
CryptoStreamMode.Write);
        byte[] byteBuffer = new byte[100];
        long nTotalByteInput = inStream.Length, nTotalByteWritten = 0;
        int nCurReadLen = 0;
        while (nTotalByteWritten < nTotalByteInput)
        {
           nCurReadLen = inStream.Read(byteBuffer, 0, byteBuffer.Length);
           myEncryptStream.Write(byteBuffer, 0, nCurReadLen);
           nTotalByteWritten += nCurReadLen;
        }
}
```

4.3 Các thuật toán mã hóa bất đối xứng trong .NET :

.NET Framework thực thi thuật toán mã hóa bất đối xứng thông qua lớp cơ sở **AsymmetricAlgorithm** cũng giống như việc sử dụng các thuật toán mã hóa đối xứng thông qua lớp **SymmetricAlgorithm**. Sau đây là sơ đồ các lớp trong lớp **AsymmetricAlgorithm** :



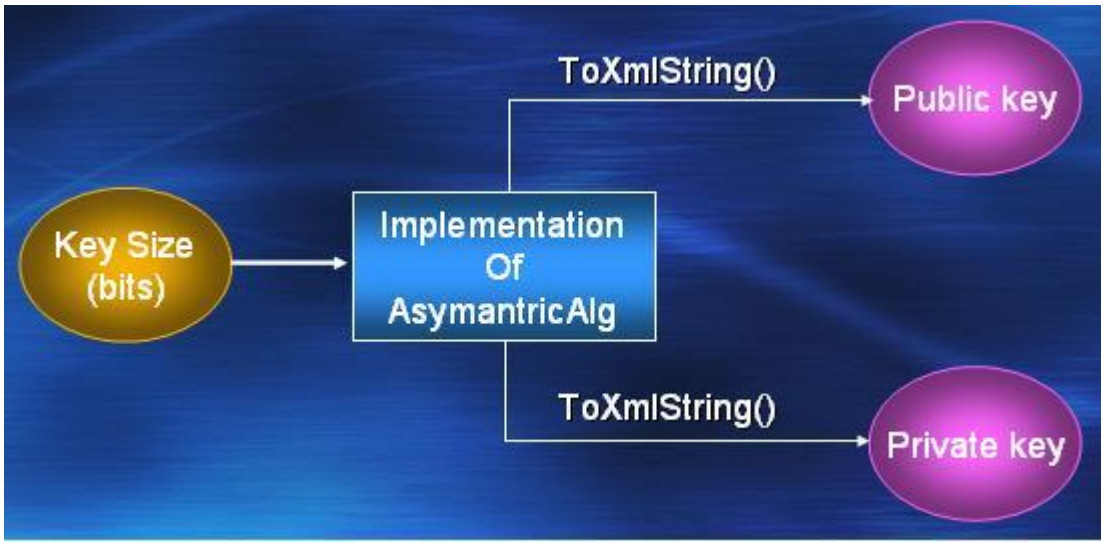
Hình 4.3 : Thuật toán mã hóa bất đối xứng trong lớp SymmetricAlgorithm

Bảng các phương thức trong lớp **AsymmetricAlgorithm** :

Phương thức chung	**Ý nghĩa**
KeySize	Kích thước của khóa tính theo bits
LegalKeySizes	Giá trị kích thước khóa hợp lệ tính theo byte của thuật toán mã hóa bất đối xứng hiện tại
KeyExchangeAlgorithm	Chỉ định thuật toán trao đổi khóa được sử dụng và cách thức trao đổi khóa công khai và khóa bí mật
SignatureAlgorithm	Chỉ định tên thuật toán được sử dụng để ký trên đối tượng hiện thời
FromXmlString()	Tái tạo lại đối tượng thuật toán mã hóa bất đối xứng từ 1 file XML
ToXmlString()	Trả về một thể hiện XML cho đối tượng bất đối xứng đang sử dụng

Chúng ta cùng xem một ví dụ về cách sử dụng thuật toán mã hóa bất đối xứng trong .NET :

-Sinh cặp khóa công khai và khóa bí mật :




```
static void Generatakey(string szKeyName, int nKeySize)
{
    try
    {
     RSACryptoServiceProvider myRSA = new
RSACryptoServiceProvider(nKeySize);
     myRSA.PersistKeyInCsp = false;

     string szPrivkey, szPubkey;
     szPrivkey = myRSA.ToXmlString(true);
     szPubkey = myRSA.ToXmlString(false);

     FileStream fsPub = new FileStream(szKeyName+"Pub.key",
FileMode.Create,FileAccess.Write);
     byte[] bytePubkey = ASCIIEncoding.ASCII.GetBytes(szPubkey);
     fsPub.Write(bytePubkey, 0, bytePubkey.Length);
```

fsPub.Close();

FileStream fsPriv = new FileStream(szKeyName+"Priv.key",FileMode.Create, FileAccess.Write);

byte[] bytePrivkey = ASCIIEncoding.ASCII.GetBytes(szPrivkey);

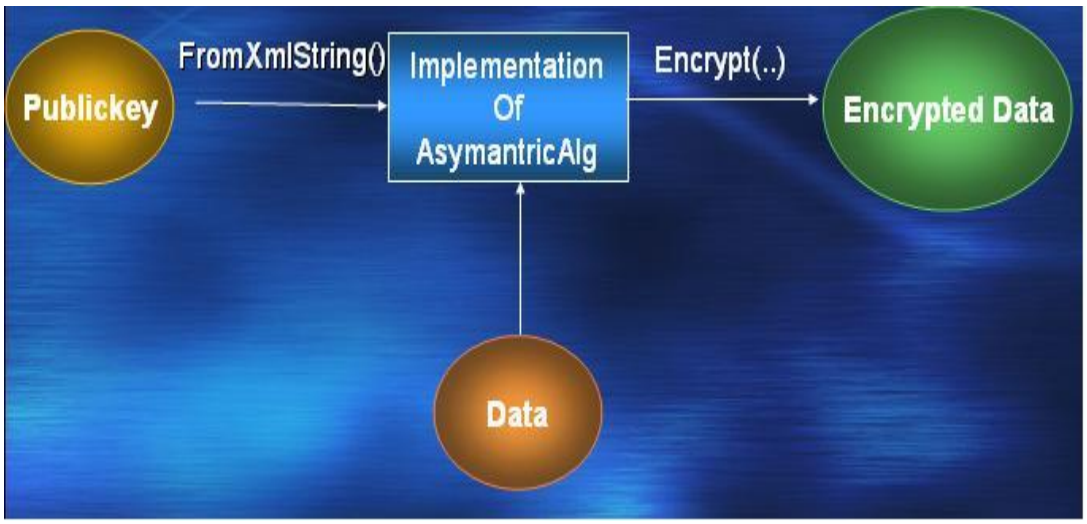
fsPriv.Write(bytePrivkey, 0, bytePrivkey.Length);

fsPriv.Close();

} catch { }

}

-Mã hóa với khóa công khai



static void EncryptFile(string szFileInput, string szFileEnc, string szPubKey)

{

try

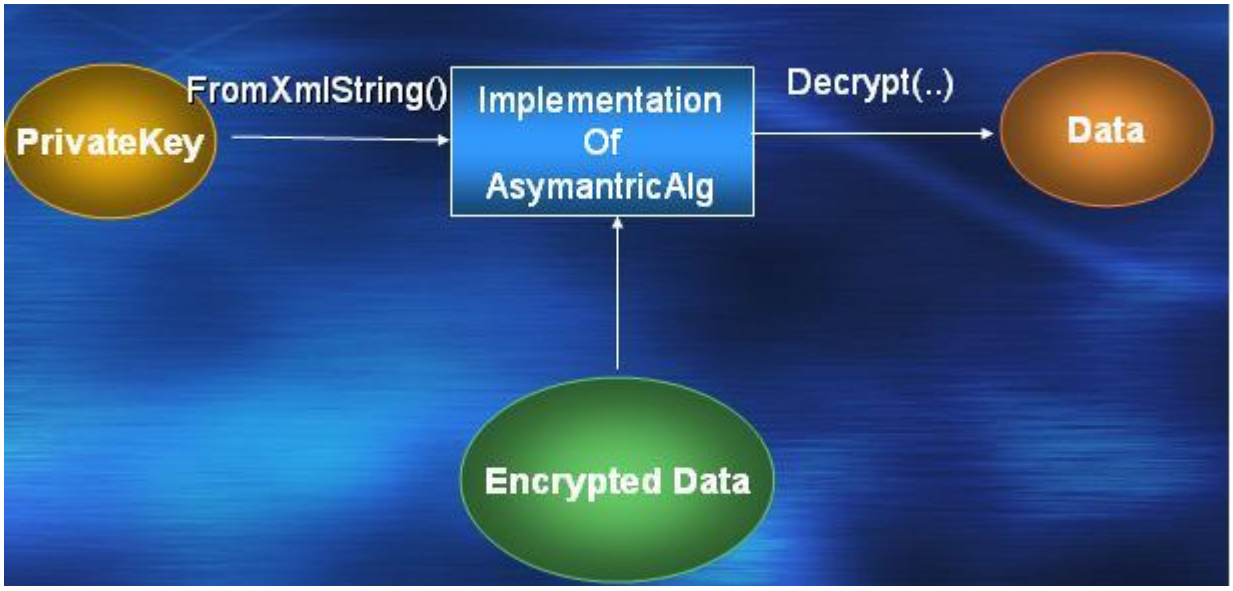
{

FileStream fsPubkey = new FileStream(szPubKey, FileMode.Open, FileAccess.Read);

FileStream fsInput = new FileStream(szFileInput, FileMode.Open, FileAccess.Read);

```
        FileStream fsOutput = new FileStream(szFileEnc, FileMode.Create,
FileAccess.Write);
        RSACryptoServiceProvider myRSA = new
RSACryptoServiceProvider();
        byte[] bytePubkey = new byte[fsPubkey.Length];
        fsPubkey.Read(bytePubkey, 0, bytePubkey.Length);
        myRSA.FromXmlString(ASCIIEncoding.ASCII.GetString(bytePubkey,
0, bytePubkey.Length));
        byte[] byteInput = new byte[fsInput.Length];
        fsInput.Read(byteInput, 0, byteInput.Length);
        byte[] byteEnc;
        byteEnc = myRSA.Encrypt(byteInput, false);
        fsOutput.Write(byteEnc, 0, byteEnc.Length);
        fsInput.Close();
        fsPubkey.Close();
        fsOutput.Close();
        }
        catch (CryptographicException ex) {Console.WriteLine(ex.Message);}
}
```

-Giải mã với khóa bí mật :




```
static void DecryptFile(string szFileInput, string szFileEnc, string szPrivKey)
{
    try
    {
    FileStream fsPrivkey = new FileStream(szPrivKey, FileMode.Open,
FileAccess.Read);
    FileStream fsInput = new FileStream(szFileInput, FileMode.Open,
FileAccess.Read);
    FileStream fsOutput = new FileStream(szFileEnc, FileMode.Create,
FileAccess.Write);
    RSACryptoServiceProvider myRSA = new
RSACryptoServiceProvider();
    byte[] bytePrivkey = new byte[fsPrivkey.Length];
    fsPrivkey.Read(bytePrivkey, 0, bytePrivkey.Length);
    myRSA.FromXmlString(ASCIIEncoding.ASCII.GetString(bytePrivkey,
0, bytePrivkey.Length));
    byte[] byteInput = new byte[fsInput.Length];
    fsInput.Read(byteInput, 0, byteInput.Length);
```

```
        byte[] byteDec;
        byteDec = myRSA.Decrypt(byteInput, false);
        fsOutput.Write(byteDec, 0, byteDec.Length);
        fsInput.Close();
        fsPrivkey.Close();
        fsOutput.Close();
        }
        catch (CryptographicException ex)     {
Console.WriteLine(ex.Message);}
}
```

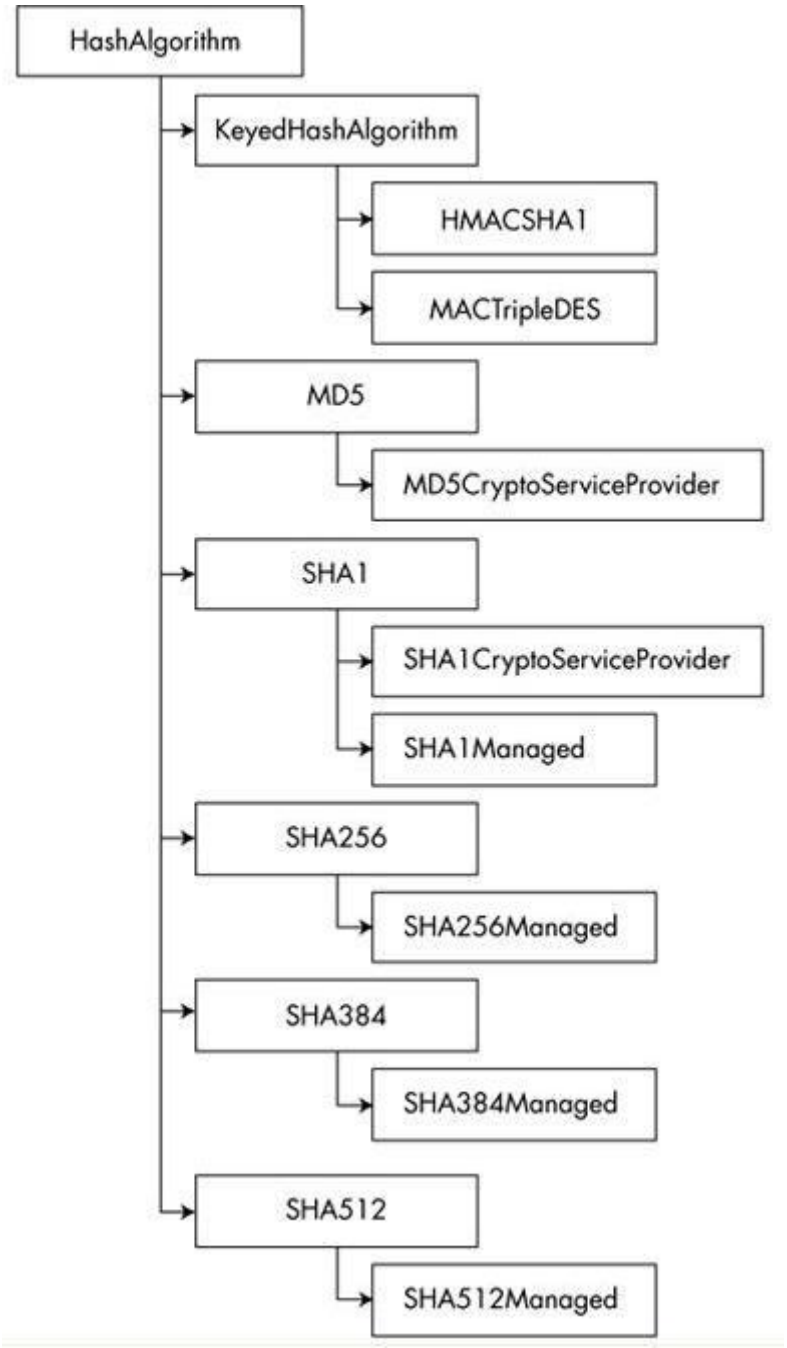
4.4 Các thuật toán hàm băm trong .NET Framework :

Có 2 loại thuật toán hàm băng thường dùng là SHA-1 (Secure Hash Algorithm được đưa ra bởi NIST vào giữa những năm 1990) và MD5 (Thuật toán Message Digest đưa ra bởi R.Rivest trong những năm đầu 1990). Thêm nữa, một vài phiên bản mới của SHA đã được công bố. Giải thuật băm có khóa cũng rất quan trọng trong việc xác thực thông tin. Tất cả những thứ đó đều được hỗ trợ bởi .NET Framework dưới dặng các lớp trong **HashAlgorithm**.

- **MD5**
- **SHA-1**
- **SHA-256**
- **SHA-384**
- **SHA-512**
- **KeyedHashAlgorithm**

Dưới đây là sơ đồ lớp trong lớp **HashAlgorithm**. Nó cung cấp các lớp trìu tượng khác như: **KeyedHashAlgorithm, MD5, SHA1, SHA256, SHA384, SHA512** những thuật toán mã hóa hàm băm thường dùng ngày nay. Các lớp này

cũng là các lớp trìu tượng nên ta không thể thao tác trực tiếp lên lớp. Ứng với mỗi lớp sẽ có một lớp thực thi cụ thể và ta sẽ sử dụng chúng trực tiếp.



Hình 4.3 : Thuật toán băm trong .NET

Lớp **MHACSHA1** cung cấp hàm băm có khóa sử dụng SHA-1 làm hàm băm. Lớp **MACTripleDES** cung cấp băm có khóa sử dụng thuật toán mã hóa Triple DES làm hàm băm. HMAC gần giống với chữ ký số, dùng để xác thực văn bản và tính toàn vẹn, tuy nhiên nó khác ở chỗ nó không phải một bộ dùng để kiểm tra và từ chối. Trái với chữ ký số sử dụng lược đồ bất đối xứng với khóa bí mật, HMAC sử dụng lược đồ đối xứng cả nơi nhận và nơi gửi sẽ có cùng một khóa để chấp nhận và thẩm tra MAC. Khi mà khóa được biết bởi nhiều hơn 1 cá nhân, sẽ không có điều j có thể chứng minh được là có một cá nhân thứ 3 sẽ được chấp nhận bởi thuật toán băm. Do đó MACs nhỏ hơn và ít an toàn hơn so với chữ ký số.

Mặc dù .NET Framework có khả năng mở rộng cao, có thể chấp nhận một số thuật toán băm của cá nhân như một lớp trong **HashAlgorithm**, nhưng điều quan trọng là bạn không nên thử thiết kế một thuật toán của riêng mình. Xét về vấn đề bảo mật, một thuật toán mã hóa phải được kiểm tra bởi rất nhiều chuyên gia trong một thời gian dài. Phần lớn người sử dụng thường sử dụng các thuật toán đã tồn tại và được tin tưởng trong một thời gian dài như MD5 và SHA.

Ngoài một số thuật toán được hỗ trợ trong .NET, không còn một thuật toán hàm băm nào được khuyến cáo sử dụng trong nghành công nghiệp máy tính. **Windows CrytoAPI** của Microsoft có hỗ trợ 3 thuật toán băm là **MD4, MD5** và **SHA-1**. .NET Framework không hỗ trợ **MD4** từ khi có **MD5, MD5** phát triển từ **MD4** khi mà phần lớn các lỗi đã được sửa. .NET Framework hỗ trợ **SHA-256, SHA-384, SHA-512** những thuật toán mà **CrytoAPI** không hỗ trợ

4.4.1 Lớp HashAlgorithm :

Lớp **HashAlgorithm** có thuộc tính chung **Hash** có dạng mảng byte chứa các giá trị của bảng băm. Thuộc tính chung **HashSize** lấy kích cỡ của mã băm dưới dạng các bit. Phương thức chung quan trọng nhất trong **HashAlgorithm** là **ComputerHash**, nó sẽ biến các giá trị đầu vào thành chuỗi byte. Dưới đây là một ví dụ về cách sử dụng lớp **HashAlgorithm** để mã

hóa theo thuật toán **SHA-1** với giá trị đầu vào là *messageByteArray* có kiểu mảng byte:

```
HashAlgorithm sha1 = new SHA1CryptoServiceProvider();
byte[] sha1Hash = sha1.ComputeHash(messageByteArray);
```

4.4.2 Lớp MD5 và SHA :

Lớp **MD5** mã hóa theo thuật toán **MD5** chấp nhận đầu vào có độ dài bất kỳ và cho kết quả có độ dài 128 bit. Ban đầu thuật toán được thiết kế để sử dụng trong các ứng dụng chữ ký số, lúc đó hàm băm được mã hóa bằng khóa riêng trong **RSA**. Thuật toán **MD5** được mở rộng từ **MD4**, nó chậm hơn nhưng lại an toàn hơn **MD4.**

Lớp **SHA1** mã hóa theo thuật toán **SHA-1**. Dữ liệu đầu vào có thế dài đến 264 bit và cho kết quả có độ dài 160 bit. SHA có thể sử dụng trong **DSA** chúng ta sẽ cùng tìm hiểu sau. **SHA-1** được phân loại vào chuẩn **NIST**- một loại chẩn về an ninh.

Lớp **SHA256**, **SHA384**, **SHA512** liên hệ mật thiết với nhau và cùng sử dụng thuật toán băm nhưng khác ở chỗ là kết quả đầu ra lần lượt là 256, 384, 512 bit. Khi bị tấn công, các thuật toán mã hóa đối xứng sẽ bị tấn công theo thỉ lệ 2^n nhưng các thuật toán hàm băm chỉ có $2^{n/2}$. Vì vậy, việc tấn công mã hóa theo kiểu hàm băm chỉ có thể thành công khi 2 dữ liệu đầu vào được tìm thấy ở cùng một giá trị đầu ra. Vì vậy việc tấn công thuật toán **SHA-160** chỉ thành công khi tấn công 2^{80} lần. Tuy thuật toán này khá an toàn nhưng không thể an toàn trong một thời gian dài.

4.4.3 Lớp KeyedHashAlgorithm

Lớp **KeyedHashAlgorithm** là một dạng biến đổi dựa trên thuật toán hàm băm, nó tính toán đầu ra của hàm băm dựa trên cả dữ liệu đưa vào nó và một phần dữ liệu được sử dụng để làm khóa của dữ liệu đầu vào. Vì thế nên thuật toán mã băm có khóa có 2 loại phương thức là *key-dependent* và *one-way*. Nó thường đc sử dụng với mục đích xác thực, khi một người biết chính xác khóa chính họ có thể chấp nhận hoặc thẩm tra hàm băm. Chính vì vậy

thuật toán hàm băm có khóa cung cấp cả phương pháp xác định tính toàn vẹn và tính xác thực để kiểm tra độ tin cậy khi trao đổi thông tin khóa. Lớp **KeyedHashAlgorithm** là một lớp trìu tượng, nó thực thi cụ thể dựa trên lớp **HMACSHA1** và **MACTripleDES**. Nhữn lớp này mã hóa thuật toán hàm băm có khóa dựa trên thuật toán **SHA-1** và **TripleDES**

4.4.4 Định danh đối tượng

Đôi khi người lập trình cần một thỏa thuận đặt tên chung để hàng trăm các chuẩn, thuật toán, và kiểu dữ liệu được được đặt dưới 1 tên duy nhất. OIDs (Object Identifiers) sẽ định nghĩa và quản lý bằng số các cấu trúc, bao gồm cả ANSI (American National Standards Institute), với mục đích định danh duy nhất các kiểu thông qua một sơ đồ phân cấp logic. Có số lượng lớn OIDs để định danh mỗi kiểu riêng biệt như : giao thức, thuật toán, các kiểu dữ liệu. Chi tiết hơn, phần lớn các thuật toán mã hóa được công nhận bởi ANSI đều được đăng ký với một tên duy nhất trong OID. Ví dụ OIDs thường được sử nhất dụng cho thuật toán băm ở hình dưới. Chúng ta sẽ xem OIDs một cách cụ thể trong các lớp của .NET Security Framework như **SignHash** và **VerifyHash** trong lớp **RSACryptoServiceProvider** và **DSACryptoServiceProvider**.

Cryptographic Hash Algorithm	**OID**
MD5	1.2.840.113549.2.5
SHA-1	1.3.14.3.2.26
SHA-256	2.16.840.1.101.3.4.2.1
SHA-384	2.16.840.1.101.3.4.2.2
SHA-512	2.16.840.1.101.3.4.2.3

Hình 4.4 : OIDs thường được sử nhất dụng cho thuật toán băm

Đoạn mã nhỏ dưới đây là ví dụ minh họa việc sử dụng OID như một thành phần trong phương thức **SignHash** của lớp **RSACryptoServiceProvider**. Tất nhiên nó chỉ là giả định vì biến **hashbytes** là một mảng byte đã được tạo ra bằng cách gọi hàm **ComputerHash** của lớp **SHA1**.

```
//create  RSA object using default key
RSACryptoServiceProvider rsa =   new RSACryptoServiceProvider();

//sign hash using OID for SHA-1
signaturebytes =   rsa.SignHash(hashbytes, "1.3.14.3.2.26");
```

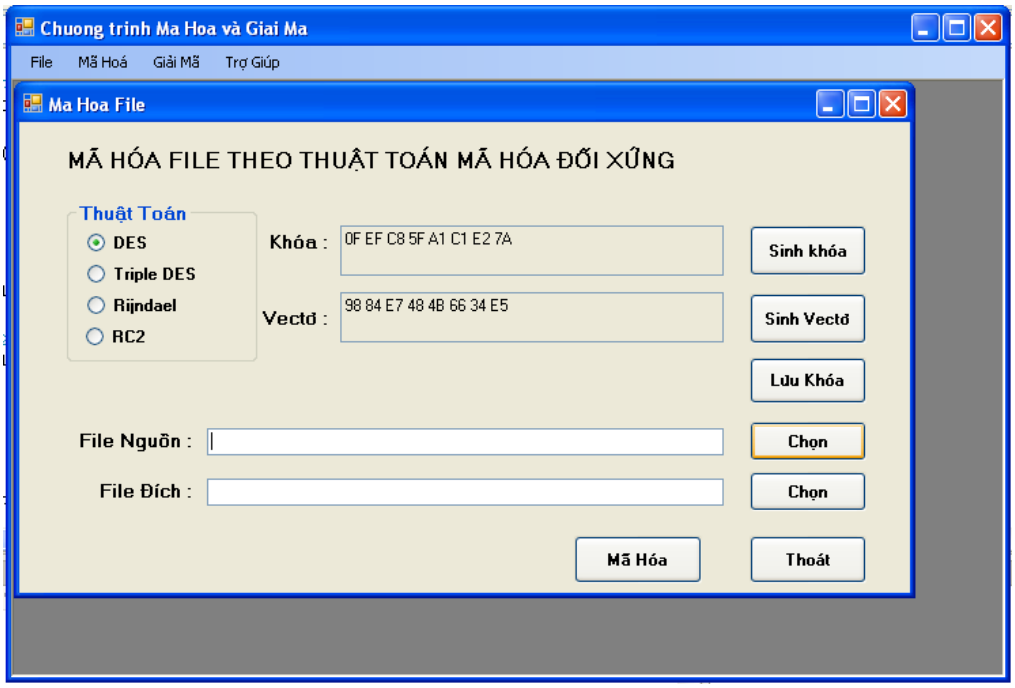
CHƯƠNG 5: LẬP TRÌNH VỚI MÃ HOÁ ĐỐI XỨNG VÀ MÃ HOÁ BẤT ĐỐI XỨNG TRONG .NET

Trong phần này chúng ta sẽ cùng đi sâu và nghiên cứu về lập trình ứng dụng mã hóa file sử dụng các thuật toán mã hóa đối xứng và bất đối xứng đã đề cập ở trên để có một cái nhìn cụ thể hơn về lớp cryptography va lập trình mã hóa trong .NET

5.1 Lập trình mã hóa đối xứng trong .NET:

5.1.1 Mã hóa file với thuật toán mã hóa đối xứng:

Chúng ta sẽ cùng xem chương trình demo và cùng giải thích làm thể nào để mã hóa một file với các thuật toán mã hóa đối xứng. Dưới đây là một số giao diện của chương trình.



-Giao diện chương trình có GroupBox bao gồm 4 loại mã hóa và 4 Radiobutton đại diện cho 4 loại mã đó. Với giá trị mặc định là mã Des được

chọn. Khi load form hoặc có sự thay đổi ở Radiobutton thì sẽ gọi đến hàm *GenKey*() và *GenIV()* để sinh khóa và và vecto cho mỗi thuật toán.

```
private void Madoixung_Load(object sender, EventArgs e)
    {
       GenKey();
       GenIV();
    }

private void radioDES_CheckedChanged(object sender, EventArgs e)
    {
       GenKey();
       GenIV();
    }

private void radioTripleDES_CheckedChanged(object sender, EventArgs e)
    {
       GenKey();
       GenIV();
    }

private void radioRijndael_CheckedChanged(object sender, EventArgs e)
    {
       GenKey();
       GenIV();
    }

private void radioRC2_CheckedChanged(object sender, EventArgs e)
    {
```

```
    GenKey();
    GenIV();
}
```

-Hàm *GenKey()* và hàm *GenIV()* sẽ dựa vào thuật toán được lựa chọn để sinh khóa và véctơ khởi tạo.

```
private void GenKey()
{
    //Generate new random IV
    SymmetricAlgorithm sa = CreateSymmetricAlgorithm();
    sa.GenerateKey();
    Key = sa.Key;

    //do UI stuff

    UpdateKeyTextBox();
}

private void GenIV()
{
    SymmetricAlgorithm sa = CreateSymmetricAlgorithm();
    sa.GenerateIV();
    IV = sa.IV;
    UpdateIVTextBox();
}
```

-Để nhận biết xem thuật toán nào được sử dụng để sinh khóa và véctơ được chính xác chúng ta dung hàm *CreateSymmetricAlgorithm()*

```
SymmetricAlgorithm CreateSymmetricAlgorithm()
    {
      //create new instance of symmetric algorithm
      if (radioRC2.Checked == true)
        return RC2.Create();
      if (radioRijndael.Checked == true)
        return Rijndael.Create();
      if (radioDES.Checked == true)
        return DES.Create();
      if (radioTripleDES.Checked == true)
        return TripleDES.Create();
      return null;
    }
```

-Ngoài ra giao điện chương trình còn có textbox khóa và textbox véctơ để hiển thị thông tin về khóa và véctơ dưới dạng hexa. Trong hàm *GenKey()* hay *GenIV()* có gọi đến hàm *UpdateKeyTextBox()* và *UpdateIVTextBox()* để cập nhật thông tin khi có sự thay đổi về khóa hay véctơ

```
private void UpdateKeyTextBox()
    {
      StringBuilder sb = new StringBuilder();
      for (int i = 0; i < Key.Length; i++)
      {
        sb.Append(String.Format("{0:X2} ", Key[i]));
      }
      txtkhoa.Text = sb.ToString();
    }
```

```
private void UpdateIVTextBox()
    {
       StringBuilder sb = new StringBuilder();
       for (int i = 0; i < IV.Length; i++)
       {
          sb.Append(String.Format("{0:X2} ", IV[i]));
       }
       txtiv.Text = sb.ToString();
    }
```

-Các Button sinh khóa và sinh véctơ sẽ gọi trực tiếp đến hàm *GenKey()* và *GenIV()* để sinh khóa. Sau đó khóa sẽ được cập nhật vào textbox

```
private void btnsinhkhoa_Click(object sender, EventArgs e)
    {
       GenKey();
    }

private void btn_sinhiv_Click(object sender, EventArgs e)
    {
       GenIV();
    }
```

-Button "Lưu khóa" sẽ lưu khóa và IV thành 2 file .key và .iv để tiện cho việc giải mã

```
private void btnluukhoa_Click(object sender, EventArgs e)
    {
```

```
        SaveFileDialog saveKeyFile = new SaveFileDialog();
        SaveFileDialog saveIVFile = new SaveFileDialog();
        saveIVFile.Filter = "Vecto File (*.iv)|*.iv";
        saveKeyFile.Filter = "Key File (*.key)|*.key";
        if (saveKeyFile.ShowDialog() == DialogResult.OK)
        {
            FileStream fsFileKey = new FileStream(saveKeyFile.FileName,
FileMode.Create, FileAccess.Write);
            byte[] keybytes = new Byte[Key.Length];

            fsFileKey.Write(Key, 0, keybytes.Length);
            fsFileKey.Close();

            if (saveIVFile.ShowDialog() == DialogResult.OK)
            {
                FileStream fsFileIV = new FileStream(saveIVFile.FileName,
FileMode.Create, FileAccess.Write);
                byte[] ivbytes = new Byte[IV.Length];

                fsFileIV.Write(IV, 0, ivbytes.Length);
                fsFileIV.Close();
            }
        }
    }
```

-TextBox File nguồn và File đích để hiển thị đường dẫn đến file nguồn và file đích. Người dùng có thể chọn đường dẫn bằng cách gõ trực tiếp hoặc sử dụng 2 Button chọn nguồn và chọn đích có sẵn

```
private void btnchonnguon_Click(object sender, EventArgs e)
    {
        OpenFileDialog openFile = new OpenFileDialog();
        openFile.Filter = "All files (*.*)|*.*";

        if (openFile.ShowDialog() == DialogResult.OK)
        {
            txtnguon.Text = openFile.FileName;
        }
    }

private void btnchondich_Click(object sender, EventArgs e)
    {
        SaveFileDialog saveencryptFile = new SaveFileDialog();
        saveencryptFile.Filter = "Encrypt files (*.enc)|*.enc";
        if (saveencryptFile.ShowDialog() == DialogResult.OK)
        {
            txtdich.Text = saveencryptFile.FileName;
        }
    }
```

Sau khi đã chọn File nguồn, File đích, thuật toán, key và véctơ người dùng có thể thực hiện việc mã hóa dễ dàng thông qua Button Mã Hóa

```
private void btnencrypt_Click(object sender, EventArgs e)
    {
        FileStream fsFileOut = new FileStream(txtdich.Text,
FileMode.Create, FileAccess.Write);
```

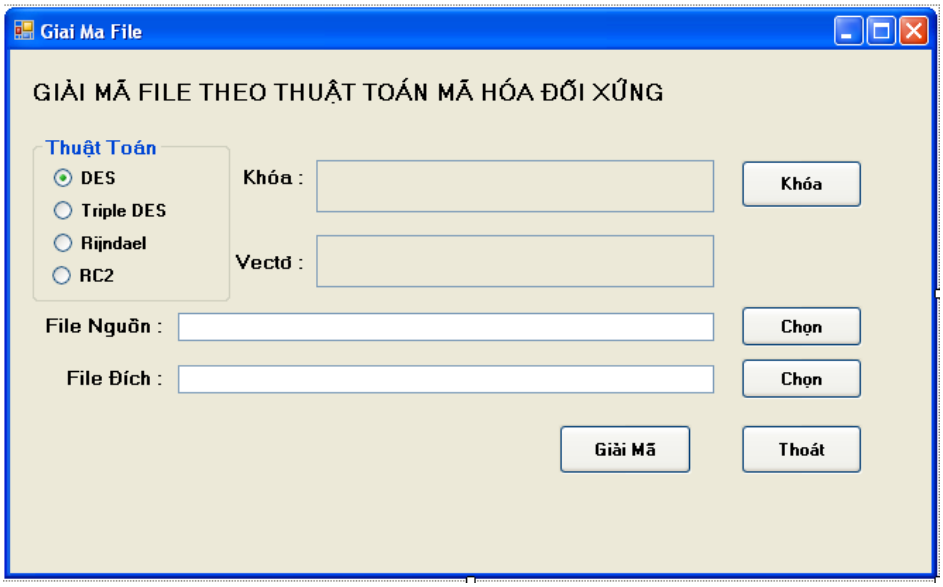
```
        FileStream fsFileIn = new FileStream(txtnguon.Text, FileMode.Open,
FileAccess.Read);

        SymmetricAlgorithm sa = CreateSymmetricAlgorithm();

        sa.Key = Key;
        sa.IV = IV;

        MemoryStream ms = new MemoryStream();
        CryptoStream csEncrypt = new CryptoStream( ms,
sa.CreateEncryptor(), CryptoStreamMode.Write);

        byte[] ByteIn = new Byte[fsFileIn.Length];
        fsFileIn.Read(ByteIn, 0, ByteIn.Length);
        fsFileIn.Close();

        csEncrypt.Write(ByteIn, 0, ByteIn.Length);
        csEncrypt.Close();
        cipherbytes = ms.ToArray();
        ms.Close();

        fsFileOut.Write(cipherbytes, 0, cipherbytes.Length);
        fsFileOut.Close();

    }
```

Chúng ta sẽ cùng đi chi tiết hơn nữa vào mã nguồn trong nút Mã Hóa để hiểu rõ hơn về quá trình mã hóa. Đầu tiên sẽ tạo ra FileStream Input với thuộc tính Open-Read để mở và đọc dữ liệu chứa trong file nguồn. FileStream

Output với thuộc tính Create-Wirte để tạo ra file mới và ghi dữ liệu lên file. Tiếp theo chúng ta xác định thuật toán đã lựa chọn và sinh thuật toán sau đó gán Key và Véc tơ của thuật toán bằng Key và Vécto đã được sinh ra trước đó. MemoryStream dùng để làm bộ nhớ đệm cho quá trình mã hóa. CryptoStream sẽ thực hiện việc mã hóa. Chúng ta cần đọc file nguồn dưới dạng các Byte cho quá trình mã hóa. Sau đó sắp xếp các Byte lại thành 1 mảng rồi ghi lên file đích. Như vậy sẽ tạo ra được file đích có chứa thông tin mã hóa.

5.1.2 Giải mã file với thuật toán mã hóa đối xứng:

Chúng ta cùng xem giao diện của Form giải mã



Giao diện và mã nguồn của một số các thành phần vẫn giống như Form mã hóa. Chúng ta chỉ xem xét đến các thành phần khác biệt trong Form giải mã.

Trước hết là Button Khóa. Dùng để đưa khóa và véctơ đã lưu trong quá trình mã hóa vào làm Khóa và Vectơ cho thuật toán đã lựa chọn. Sau đó giá trị của Khóa và Véctơ sẽ được hiển thị trên các textbox tương ứng

```
private void btnkhoa_Click(object sender, EventArgs e)
    {
        OpenFileDialog openkey = new OpenFileDialog();
        OpenFileDialog openIV = new OpenFileDialog();
        openkey.Filter = "Key File (*.key)|*.key";
        openIV.Filter = "Vecto File (*.iv)|*.iv";

        if (openkey.ShowDialog() == DialogResult.OK)
        {
            FileStream fsFileKey = new FileStream(openkey.FileName,
FileMode.Open, FileAccess.Read);
            byte[] keybytes = new Byte[fsFileKey.Length];

            fsFileKey.Read(keybytes, 0, keybytes.Length);
            fsFileKey.Close();
            Key = keybytes;
            UpdateKeyTextBox();

            if (openIV.ShowDialog() == DialogResult.OK)
            {
                FileStream fsFileIV = new FileStream(openIV.FileName,
FileMode.Open, FileAccess.Read);
                byte[] IVbytes = new Byte[fsFileIV.Length];

                fsFileIV.Read(IVbytes, 0, IVbytes.Length);
                fsFileIV.Close();
                IV = IVbytes;
                UpdateIVTextBox();
            }
```

```
        }
            }
```

Cùng đi sâu vào mã nguồn của Button Giải Mã :

```
private void btndecrpt_Click(object sender, EventArgs e)
    {
        SymmetricAlgorithm sa = CreateSymmetricAlgorithm();
        sa.Key = Key;
        sa.IV = IV;

        FileStream fsFileOut = new FileStream(txtdich.Text,
FileMode.Create, FileAccess.Write);
        FileStream fsFileIn = new FileStream(txtnguon.Text, FileMode.Open,
FileAccess.Read);



        byte[] cipherbytes = new Byte[fsFileIn.Length];
        fsFileIn.Read(cipherbytes, 0, cipherbytes.Length);
        fsFileIn.Close();

        MemoryStream ms = new MemoryStream(cipherbytes);
        CryptoStream csDecrypt = new CryptoStream(ms,
sa.CreateDecryptor(), CryptoStreamMode.Read);

        byte[] plainbytes = new Byte[cipherbytes.Length];
        csDecrypt.Read(plainbytes, 0, plainbytes.Length);
        csDecrypt.Close();
        ms.Close();
```

```
        fsFileOut.Write(plainbytes, 0, plainbytes.Length);
        fsFileOut.Close();
        fsFileIn.Close();

    }
```

Cũng giống như Button Mã Hóa nhưng đầu tiên chúng ta phải đọc file đích ở dạng các Byte. Sau đó đưa vào trong bộ đệm, cuối cùng mới giải mã. Tiếp đến lưu các Byte đầu ra vào một mảng Byte trung gian rồi ghi kết quả lên File Đích. Chúng ta sẽ có File giải mã có nội dung giống như File ban đầu.

5.1.3 Cryptogphaph Stream:

Common Language Runtime (CLR) hỗ trợ Stream có hướng, được thiết kế cho việc xử lý mã hóa. Lớp thực hiện công việc đóng gói một stream mã hóa có tên là **CryptoStream**. Mỗi quá trình mã hóa sẽ cung cấp một đối tượng **CryptoStream** để có thể kết nối đến các **CryptoStream** khác. Bằng cách nối các chuỗi mã hóa lại với nhau, đầu ra từ một đối tượng sẽ là đầu vào của đối tượng tiếp theo mà không cần phải lưu trữ riêng rẽ và đưa vào sau này.

CryptoStream là một lớp dễ dàng sử dụng, nó cho phép bạn đọc và ghi dữ liệu thông qua đối tượng chuỗi mã hóa, điều đó đồng nghĩa với bạn có thể áp dụng cho dữ liệu đầu vào đầu ra đơn giản của một file hay socket. Lớp **CryptoStream** có thể mã hóa (ở chế độ ghi) và giải mã (ở chế độ đọc) ngay tức thì. Trong chương trình trên chúng ta đã sử dụng lớp MemoryStream để kích hoạt quá trình xử lý đầu vào/đầu ra để xử lý trong bộ nhớ đệm. Đây là cấu trúc của **CryptoStream**

```
public CryptoStream(
   Stream stream,
   ICryptoTransform transform,
   CryptoStreamMode mode
);
```

5.1.4 Chống lại khóa yếu:

Có một hàm trong lớp **DES** và **TripleDES** có tên là **IsWeakKey**, nó sẽ lấy chuỗi byte có chưa khóa làm biến, và trả lại giá trị kiểu Boolean. Hàm này được sử dụng để xác định rằng khóa có yếu hay không. Chúng ta biết rằng khóa yếu trong mã hóa là dễ dàng để có thể bẻ. Khóa yếu có một số các đặc trưng sau: Nếu một mẩu tin được mã hóa bằng khóa yếu, thì sau khi mã hóa tiếp một lần nữa bằng khóa đó nó sẽ cho ra kết quả của mẩu tin ban đầu.

Vì **TripleDes** được xây dựng từ **DES** nên khóa yếu với **DES** cũng đồng nghĩa với việc nó yếu với **TripleDES**. Sẽ không có khóa yếu với các thuật toán **RC2** và **Rijndael** thế nên lớp **RC2** và **Rijndael** sẽ không có hàm **IsWeakKey**.

Đã có 4 khóa của **Des** được liệt vào danh sách khóa yếu và hiển nhiên là chúng ta có thể tránh được những khóa này. Tuy nhiên cũng không đáng lo ngại nhiều lắm vì **DES** có tới 56 bit tức là 2^{56} key có thể sử dụng trong khi số lượng khóa yếu chỉ là 4 (2^2) vậy nên xác xuất để lấy phải khóa yếu là 2^{-54}. Còn 12 khóa được đưa vào danh sách nửa-yếu, nhưng số lượng đó cũng là rất ít so với 2^{56} khóa có thể lấy nên chúng ta không cần quá quan tâm đến nó.

Trong mọi trường hợp, phần lới chương trình sử dụng mã khóa bí mật được sinh một cách tự động qua đối tượng thuật toán mã hóa, nó sẽ không bao giờ cung cấp cho ta một khóa yếu. Nếu chúng ta sử dụng lời gọi hàm **GenerateKey** thì bạn hoàn toàn có thể chắc chắn rằng chúng ta đã tránh được khóa yếu. Trong trường hợp chúng ta lấy khóa theo kiểu khác như là từ CSP hoặc nguồn ở ngoài, lớp **DES** và **TripleDES** sẽ báo lỗi khi bạn cố tình sử

dụng khóa yếu. Để tránh việc này bạn hoàn toàn có thể sử dụng hàm **IsWeakKey**.

5.1.5 Tổng kết :

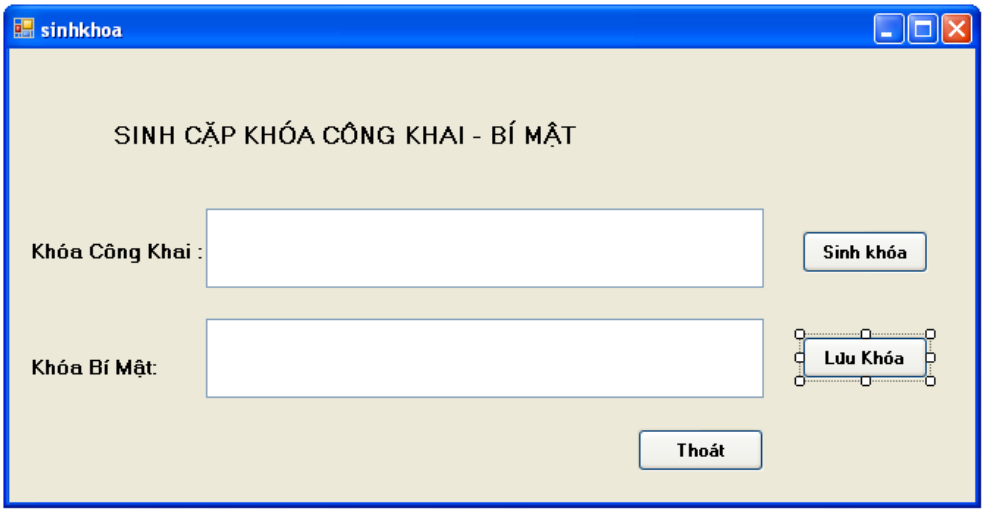
Phần đầu của chương 5 này tập trung vào thuật toán mã hóa đối xứng và các lớp .NET Framework thực thi chúng. Chúng ta tập trung vào các loại mã phổ biến như **DES**, **Triple DES**, **Rinjdael**, và **RC2** , tìm hiểu xem làm thế nào để sử dụng chúng từ lớp cơ sở **SymmetricAlgorithm** trong .Net Framework ứng dụng vào trong bảo mật thông tin. Và chúng ta cũng lướt qua ưu điểm của lớp **CryptoStream**. Tuy nhiên các vấn đề nâng cao như kiểu mã hóa (Mode) và kiểu Padding chưa được sử dụng trong chương trình.

5.2 Lập trình mã hóa bất đối xứng trong .NET:

Phần tiếp theo của chương này chúng ta sẽ cùng tìm hiểu về lập trình mã hóa bất đối xứng với 1 file trong môi trường .NET

5.2.1 Sinh cặp khóa Công khai-Bí mật :

Để thực hiện được việc mã hóa và giải mã với thuật toán mã hóa bất đối xứng chúng ta cần tạo một cặp khóa Công khai-Bí mật. Khóa công khai dùng để mã hóa dữ liệu còn khóa bí mật dùng để giải mã dữ liệu. Để sinh ra một cặp khóa phù hợp với thuật toán là cả một quá trình khó khăn với nhữn con số rất lớn. Tuy nhiên .NET đã đơn giản hóa việc sinh cặp khóa để người sử dụng không còn phải qian tâm quá nhiều đến quá trình sinh khóa. Dưới đây sẽ giới thiệu cách tạo ra cặp khóa rất đơn giản. Đây là giao diện của quá trình sinh khóa



Các textbox Khóa công khai, khóa bí mật để hiển thị các khóa. Button Sinh khóa sẽ thực hiện việc sinh khóa và hiển thị chúng trên các textbox tương ứng. Khóa sẽ được tự động sinh dưới dạng XML là định dạng mặc định.

Dưới đây là mã nguồn của Button Sinh khóa

```
private void btnsinhkhoa_Click(object sender, EventArgs e)
    {
       GenerateNewRSAParams();
    }

    private void GenerateNewRSAParams()
    {
       //establish RSA asymmetric algorithm
       RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
       publicPrivateKeyXML = rsa.ToXmlString(true);
       publicOnlyKeyXML = rsa.ToXmlString(false);
       txtPubKey.Text = publicOnlyKeyXML;
       txtPriKey.Text = publicPrivateKeyXML;
```

```
}
```

Phương thức rsa.ToXmlString() có tham số kiểu Boolean ứng với:

- true : sinh ra khóa bí mật
- false : sinh ra khóa công khai

Sau khi sinh khóa sẽ thể hiện khóa trên 2 textbox txtPrivate và txtPublic

5.2.2 Lưu khóa dưới dạng XML :

Button Lưu khóa giúp các bạn dễ dàng lưu khóa công khai và khóa bí mật thành 2 file xml

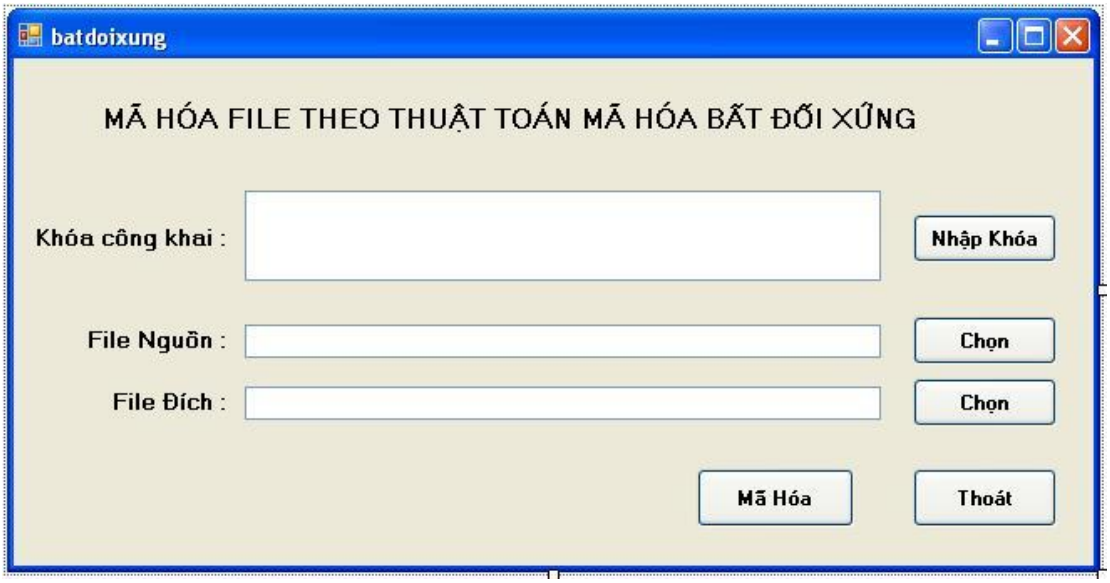
```
private void btn_luuRSA_Click(object sender, EventArgs e)
    {
        SaveFileDialog savePubKey = new SaveFileDialog();
        SaveFileDialog savePriKey = new SaveFileDialog();
        savePriKey.Filter = "PriveteKey (*.xml)|*.xml";
        savePubKey.Filter = "PublicKey (*.xml)|*.xml";
        if (savePriKey.ShowDialog() == DialogResult.OK)
        {
            StreamWriter writerPri = new
StreamWriter(savePriKey.FileName);
            writerPri.Write(publicPrivateKeyXML);
            writerPri.Close();

            if (savePubKey.ShowDialog() == DialogResult.OK)
            {
                StreamWriter writerPub = new
StreamWriter(savePubKey.FileName);
                writerPub.Write(publicOnlyKeyXML);
                writerPub.Close();
```

```
        }
    }
        }
```

Ở đây ta dùng phương thức StreamWriter để lưu khóa vào file đích dưới dạng XML

5.2.3 Mã hóa file với thuật toán mã hóa bất đối xứng:

Dưới đây là giao diện của chương trình



Các cách thức chọn File nguồn và File đích giống như phần mã hóa bất đối xứng ở trên. Chúng ta chỉ tìm hiểu về sự khác biệt về mã nguồn của phương pháp mã hóa này.

Như đã biết, mã hóa bất đối xứng cho phép chúng ta dùng khóa công khai để mã hóa dữ liệu. Vậy nên có Button Nhập khóa để những người đã có khóa công khai nhập vào và dùng nó để mã hóa dữ liệu mà không cần phải sinh khóa.

```
private void btnNhapkhoa_Click(object sender, EventArgs e)
    {
        OpenFileDialog openPubKey = new OpenFileDialog();
```

```
        openPubKey.Filter = "PublicKey (*.xml)|*.xml";

        if (openPubKey.ShowDialog() == DialogResult.OK)
        {
            StreamReader reader = new
StreamReader(openPubKey.FileName);
            string sPubkey = reader.ReadToEnd();
            txtPublic.Text = sPubkey;
            reader.Close();
        }
    }
```

Chúng ta dùng phương thức StreamReader để đọc dữ liệu từ khóa công khai và thể hiện lên text box tương ứng.

Tiếp đến chúng ta sẽ tìm hiểu quá trình mã hóa File qua Button Mã Hóa

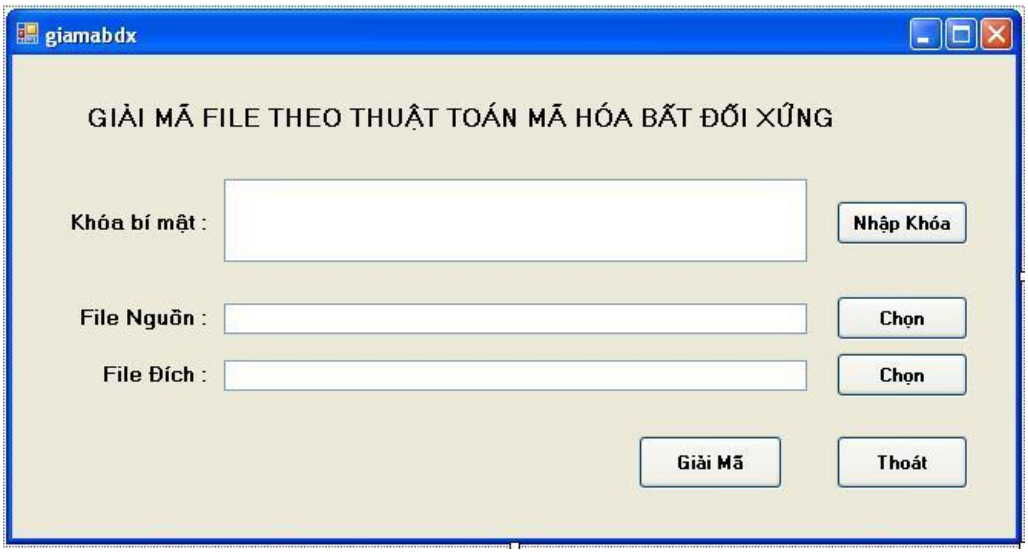
```
private void btnencrypt_Click(object sender, EventArgs e)
    {
        RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();

        rsa.FromXmlString(txtPublic.Text);

        FileStream fsFileOut = new FileStream(txtdich.Text,
FileMode.Create, FileAccess.Write);
        FileStream fsFileIn = new FileStream(txtnguon.Text, FileMode.Open,
FileAccess.Read);

        byte[] ByteIn = new Byte[fsFileIn.Length];
```

```
        fsFileIn.Read(ByteIn, 0, ByteIn.Length);
        fsFileIn.Close();
        cipherbytes = rsa.Encrypt(ByteIn, false);
        fsFileOut.Write(cipherbytes, 0, cipherbytes.Length);
        fsFileOut.Close();
    }
```

Đầu tiên là tạo ra thuật toán mã hóa RSA. Sau đó đọc khóa công khai từ textbox txtPublic để mã hóa file. Tiếp đến sẽ nhận các dữ liệu đầu vào và đầu ra dưới dạng FileStream. Ta biến đổi file đầu vào từ FileStream sang dạng các byte để dễ dàng mã hóa. Cuối cùng là mã hóa và ghi chúng lên File đích.

5.2.4 Giải mã file với thuật toán mã hóa bất đối xứng :

Phần này sẽ đi vào việc phân tích mã nguồn của quá trình giải mã file. Giao diện của trương trình như trong phần mã hóa



Phần khác nhau chính là ở đây người ta dùng khóa bí mật để giải mã. Các Button đều giống như ở phần mã hóa. Chúng ta chỉ nói về sự khác biệt duy nhất Button Giải Mã

```
private void btndecrypt_Click(object sender, EventArgs e)
    {
        RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
        rsa.FromXmlString(txtPriKey.Text);
```

```
        FileStream fsFileOut = new FileStream(txtdich.Text,
FileMode.Create, FileAccess.Write);
        FileStream fsFileIn = new FileStream(txtnguon.Text, FileMode.Open,
FileAccess.Read);

        byte[] ByteIn = new Byte[fsFileIn.Length];
        fsFileIn.Read(ByteIn, 0, ByteIn.Length);
        fsFileIn.Close();

        byte[] cipherbytes = rsa.Decrypt(ByteIn, false);

        fsFileOut.Write(cipherbytes, 0, cipherbytes.Length);
        fsFileOut.Close();
    }
```

Hoàn toàn giống với quá trình mã hóa chỉ khác một điều là quá trình giải mã sẽ sử dụng khóa bí mật và hàm Decrypt() để giải mã. Sau khi giải mã xong dữ liệu sẽ được ghi lên file đích và hoàn toàn giống với file gốc trước khi mã hóa.

5.2.5 Tổng kết :

Phần thứ 2 của chương 5 giới thiệu với chúng ta về thuật toán mã hóa bất đối xứng, cụ thể là thuật toán RSA. Qua đó ta thấy được ưu điểm của thuật toán bất đối xứng trong bài toán thực tế khi không cần phải đưa ra khóa bí mật. Và từ đó có cái nhìn tổng qua về cách thức mã hóa của thuật toán RSA để xây dựng ứng dụng vào lập trình trong .Net qua lớp **RSACryptoServiceProvider**. Cuối cùng chúng ta biết được cách lưu trữ các thành phần của thuật toán RSA thành dạng XML để có thể chia sẽ cho các đơn vị khác.

KẾT LUẬN

1 Phần làm đã làm được:

Sau quá trình nghiên cứu, thực hiện luận văn, em đã tìm hiểu và nắm được một số vấn đề như

-Tìm hiểu về một số loại mã hóa cổ điển cũng như một số loại mã hóa cao cấp được sử dụng hiện nay

-Tìm hiểu về .NET Framework cũng như .NET Framework trong bảo mật thông tin

-Tìm hiểu về lớp Cryptography trong .NET Framework hỗ trợ trong lập trình bảo mật

-Xây dựng được chương trình demo về mã hóa file với một số thuật toán mã hóa hiện đại

2 Phần chưa thực hiện được:

Bên cạnh những phần đã thực hiện được, đồ án vẫn còn tồn tại một số hạn chế như:

-Chưa tìm hiểu được về một số loại mã hóa như RC2

-Chương trình ứng dụng còn đơn giản, giao diện chưa thân thiện

-Chưa bắt được tất cả các lỗi và nêu rõ lỗi do đâu

-Chưa xây dựng hệ thống trợ giúp hỗ trợ người sử dụng

-Mã hóa RSA chưa thực hiện được với các file lớn

3 Hướng phát triển:

Qua một số những phần chưa thực hiện được.Vì thời gian hạn chế và mức độ nghiên cứu chưa sâu rộng. Em rất mong đồ án sẽ là tài liệu tham khảo

cho các bạn sinh viên khóa sau khắc phục một số khuyết điểm để chương trình được hoàn thiện hơn như:

-Xây dựng ứng dụng có giao diện than thiện dễ sử dụng

-Bắt lỗi chi tiết từng lỗi và nêu ra lỗi ở đâu

-Xây dựng hệ thống trợ giúp

-Mở rộng RSA để có thể mã hóa được các file có dung lượng lớn

4 Lời cảm ơn

Dưới sự hướng dẫn nhiệt tình của thầy giáo Thạc sĩ Lê Thụy, em đã tiếp thu được nhiều kiến thức và kinh nghiệm để có thể hoàn thành bài báo cáo

Do thời gian và kiến thức còn hạn chế nên đồ án không tránh khỏi những thiếu sót, vậy em rất mong nhận được sự nhận xét và đánh giá của các thầy cô giáo cùng ý kiến đóng góp của các bạn để dồ án em được hoàn thiện hơn

Em xin tỏ lòng biết ơn tới thầy giáo Thạc sĩ Lê Thụy đã giành thời gian và tâm huyết giúp đỡ em vượt qua những khúc mắc trong quá trình làm đồ án. Em cũng xin cảm ơn thầy cô giáo trong trường đã tạo điều kiện cho em hoàn thành đồ án. Em xin chân thành cảm ơn.

TÀI LIỆU THAM KHẢO

1 .NET Security and Cryptography

2 Mã hóa và ứng dụng - Tác giả :TS Dương Anh Đức – ThS Trần Minh Triết

3 http://dot.net.vn/

4 http://www.dotnetspider.com/

5 Và một số tài liệu trên Internet