APT – A Novell Symmetric Key Cryptographic Algorithm for Securing VPN's

G. Sudha Sadhasivam[1]

ABSTRACT

This paper proposes a novel symmetric key cryptographic algorithm – Advanced Polynomial Transformation (APT) for securing the Virtual Private Networks (VPNs). VPNs allow users to communicate and access information over the public Internet and other IP-based networks. Secure virtual connections or 'tunnels' are established to overcome the security threats in VPN. This paper focuses on the Encryption 'tunnels' by proposing a novel 96-bit symmetric key cryptographic algorithm that uses APT (Advanced Polynomial Transformation). APT is a Block-Cipher that operates on 64-bit plain-text blocks. It makes use of algebraic polynomial functional mappings for the plaintext to cipher transformation by poly-alphabetic substitution. It transforms plain text to 64-bit ciphers achieving an input-output bit conversion ratio of 1:1.

Keywords : APT, symmetric key encryption, VPN.

1. INTRODUCTION

Rapid growth of VPN [1] technology over the past few years has enabled the business organizations to extend their own networks to create a geographically distributed workplace. Dramatic cost savings can be realized when compared to the expense of building and operating a private network since VPNs utilize the public telecommunications infrastructure instead of dedicated circuits. But along with the rapid growth of the technology and cost savings the VPNs [2] are likely to suffer security breaches. The security of resources both inside and out of the VPN very much depends on the virtual private network and is always remaining private. As remote accesses and business-to-business situations involve remote nodes over which there is no adequate administrative control, the VPNs are far more likely to suffer breaches in isolation than the traditional private networks. Secure virtual connections or 'tunnels' are established to overcome the security threats in VPN. These 'tunnels' refer to the Encryption process being carried out for secure connection establishment. This paper proposes a novel 96-bit symmetric key cryptographic algorithm that uses APT (Advanced Polynomial Transformation) for encryption. It is a block-cipher which operates on 64-bit plain-text blocks. APT transforms the plaintext blocks into 64-bit ciphers achieving an input-output bit conversion ratio of 1:1.

2. PROPOSED METHOD

APT is a symmetric key crypto algorithm that uses a 96-bit key with a 64-bit key called the *base*, k_b, 16-bit key called rand1, k_1, and another 16-bit key called rand2, k_2. The algorithm takes 64-bit plain text block and encrypts it into 64-bit cipher. Both the encryption and the decryption process execute a set of computationally complex initialization procedures using the 96-bit key. These procedures generate a *base* domain and a *rand* domain each of 2^{16} half words (1 word = 32 bits). The core of the remaining encryption process rests with the mapping of the plain text with a base string using a

[1]Assistant Professor, Department of CSE,
PSG College of Technology,
Coimbatore– 641 004, Tamil Nadu, India.
E-mail : sudhasadhasivam@yahoo.com

function F_x. The function F_x is randomly chosen from a domain of 2^{64} functions. Based on the mapping a status vector is generated. The status vector further undergoes another transformation to yield the cipher. The decryption process is a logical backward retrace of the encryption process. A detailed description of the encryption and decryption mechanism is given in the following sub-sections.

2.1 Encryption

The proposed encryption process consists of initialization, exchange and transformation procedures.

2.1.1 Initialization

Fig. 1: Initialisation Algorithm

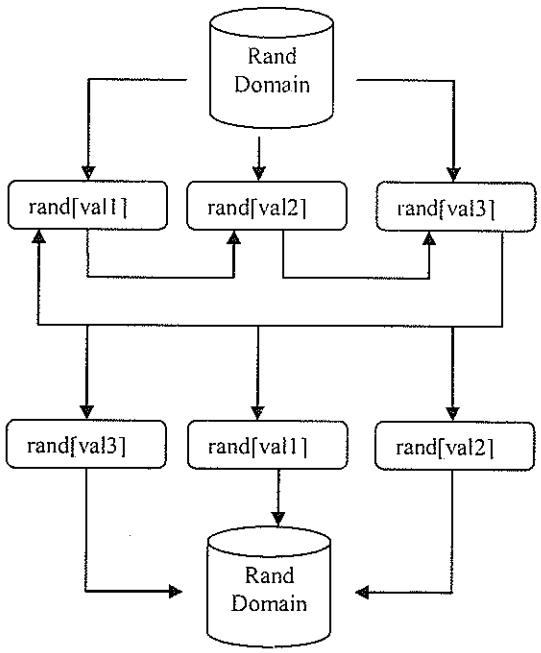
The first step in encryption is initialization (fig.1). Two 16- bit keys namely rand1 and rand2 are given to the initialization block which executes a series of C-Exchanges (fig. 3) and bit-wise operations iteratively to build two domains namely the *base* domain and the *rand* domain each of 2^{16} half words. This is a computationally complex block requiring 2^{16} iterations and it is also the source for randomness in the encryption process (fig 1) . The large size of the rand domains makes the algorithm secure against many attacks. This phase also aids to provide

```
Algorithm Initialization
{
  base1 = k_b ; rand1 = k_1; rand2 = k_2
   for index = 1 to 2^16
      rand[index] = index;
  end for
  for index = 1 to 2^16
     rand11 = function1(rand1);
     rand12 = function1(rand2);
     C-Exchange ( rand[ |rand11 – rand12| ] ,
                  rand[rand11] ,
                  rand[rand11 + rand12] );
     rand1 = rand1  XOR  rand[rand11];
     rand2 = rand2  XOR  rand[rand12];
     base[index] = rand1;
  end for
}
```

```
Algorithm function1(arg)
{
  A = XOR (1st - nibble, 2nd - nibble, 3rd - nibble);
  B = XOR (4th - nibble, 1st - nibble, 2nd - nibble);
  C = XOR (3rd - nibble, 4th - nibble, 1st - nibble);
  D = XOR (2nd - nibble, 3rd - nibble, 4th - nibble);
  return (Append (A, B, C, D));
}
Algorithm C-Exchange( A , B , C )
{
        temp = B;
        B = A;
        A = C;
        C = temp;
return;
}
```

Fig. 2: C-Exchange Procedure

unconditional security to the algorithm as discussed in section 3.The C-Exchange block takes three values A, B, C as its parameters where, A = rand[| rand11 – rand12 |], B = rand[rand11] , C = rand[rand11 + rand12]. It exchanges the values in a circular manner as shown in algorithm given in figure 2.



val1, val2, val3 are any three indexes.

Fig.3: Working of C-Exchange Algorithm

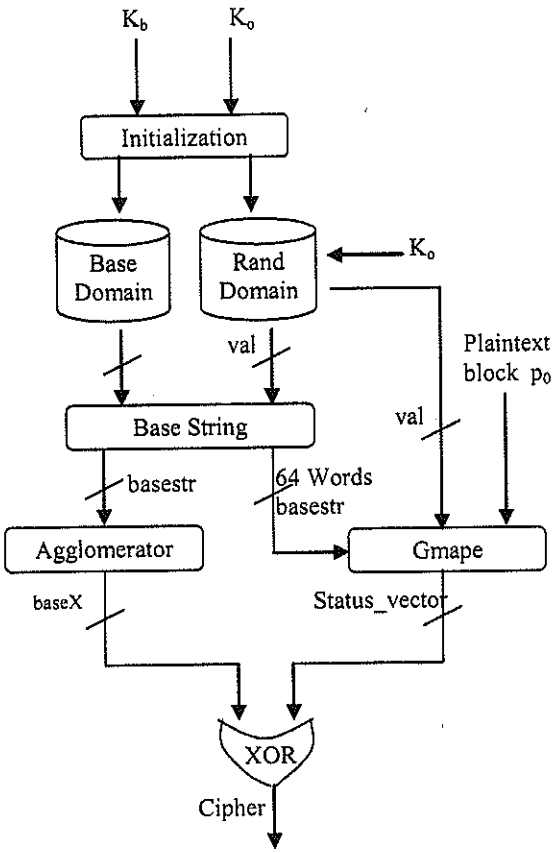


Fig. 4: Overview of Encryption

2.1.2 Transformation

The second step in the Encryption process is Transformation. The base string constructor picks up 256 half words from the large base domain randomly (the source of randomness being the rand domain) to frame the base string. The constructed base string maps the plaintext onto the cipher using a function F_x. F_x is a 64-bit function of the form $(1\ldots2^{16})x^2+(1\ldots2^{16})x+(1\ldots2^{32})$. The value of F_x is obtained from the 64-bit key, K_b. The base string basestr (256 half words), the plaintext p_0 (64 bits), a random parameter

```
Algorithm Encryption
{
    call initialization();
    ind = K1 ;
    for index = 1 to Plaintext length or Plaintext
                blocks
        val = rand[ind+index];
```

```
Algorithm Encryption
{
    call initialization();
    ind = K1 ;
    for index = 1 to Plaintext length or Plaintext
                blocks
        val = rand[ind+index];
        Basestr = base[base[val + 0]] +
                  base[base[val + 1]] +
                  base[base[val + 2]] + . . . +
                  base[base[val + 255]];
        basex = function2(basestr);
        ind = rand[basex];
        Fx = Kb;
        Kb = Kb XOR ind;
        status_vector[] = encrypt ( Basestr,
                          plaintext, Fx);
        cipher = BaseX XOR status_vector[];
    end for
}
Algorithm function2(arg)
{
    A = XOR(arg[1st word], arg[2nd word],,...,
            arg[64th word]);
    return(A);
}
Algorithm encrypt(Basestr,plaintext,Fx)
{
    for i = 1 to 64
        Index[I] = f(xi);
        Observed[I] = base[(index[I] % 4096)];
    end for
    Input[] = toBits(plaintext);
    status_vector[] = observed[] XOR input[];
    return(status_vector);
}
```

Fig. 5: Encryption Algorithm

Fx (64 bits) and basex (64 bits) are passed to encrypt (Gmape) block. The index vector is first generated from f(1:64). The observed vector is then built by traversing the base string and using the bit values at the appropriate locations as given by the index vector. The status vector is obtained by XORing the input and the observed vector. The basestr is passed to the Agglomerator, which transforms it into baseX of 64 bits. The cipher is obtained by XORing the status vector with baseX. The above steps are repeated for subsequent encryptions. A diagrammatic representation of the proposed encryption algorithm (fig 5) is shown in figure 4.

2.2 Decryption

Fig. 5: Encryption Algorithm

The decryption process follows the same suit as that of the encryption process with regards to the initialization procedure and the construction of the *base* domain and the *rand* domain. The *base string constructor* constructs a 256 half-word base string and the *Agglomerator* generates the 64-bit baseX. F_x is obtained in the same manner as in the encryption process. The generated baseX is XORed with the cipher to obtain the status vector, which is passed to the *decrypt* block. This block builds the 64-bit index vector from f (1:64). The bit values in the corresponding locations as given by the index vector yield the 64-bit observed vector. The plain text is obtained by XORing the observed vector with the status vector. The proposed decryption algorithm as shown in figure 7 is given in fig. 6.

3. APT Properties and Strengths

This section presents an analysis on the properties of the proposed encryption/decryption algorithm using APT technique. The following are some of the basic properties of the APT algorithm:

1. It achieves an input-output bit transformation ratio of 1:1.

2. Altering a key bit alters each cipher bit with probability 0.5

```
Algorithm Decryption
{
  call Initialization()
  call initialization();
  ind = K1
  for index = 1 to Plaintext length or Plaintext
              blocks
    val = rand[ind+index];
    Basestr = base[base[val + 0]] +
              base[base[val + 1]] +
              base[base[val + 2]] + ..... +
              base[base[val + 255]] ;
    Basex = function2(basestr)
    ind = rand[basex]
    Fx = Kb;
    Kb = Kb XOR ind;
    status_vector = Cipher XOR basex
    Plaintext= decrypt(basestr,status_vector,Fx
  end for
}
Algorithm decrypt(basestr,status_vector,Fx)
{
  for I = 1 to 64
    Index[I] = f(xi);
    observed[I] = base[(index[I] % 4096)];,
    Plaintext[I] = observed[I] XOR
                   status_vector[I];
  end for
  return(Plaintext)
}
```

Fig. 6: Decryption Algorithm

3. Uncondtional Security :

Since the base string used is different for every plain text block. On an average, if n^{th} plaintext block uses the 64 word base string K_r, the probability for the $(n+1)^{th}$ plaintext to use the same K_r is $1 / (2^{64} P_{64})$.

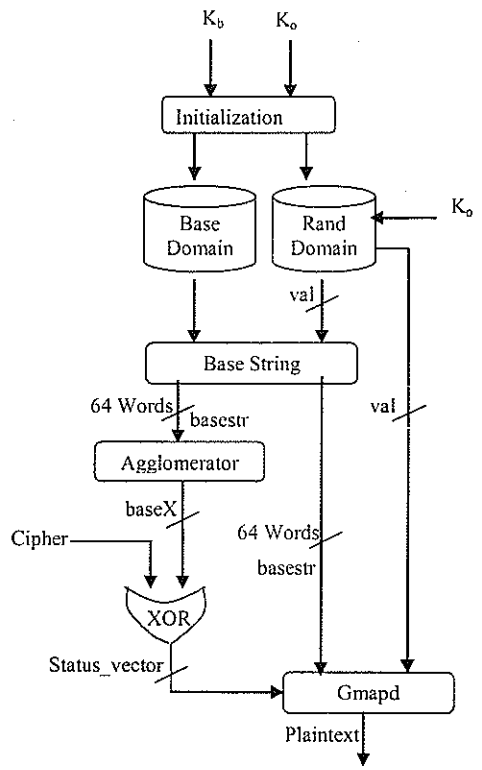


Fig. 7: Overview of Decryption Process

4. **Computational Security:**

A computationally complex procedure generates base array and completely randomizes *rand* array of size 2^{64}. Polynomial attacks are of the order $O(2^{64})$. Although it is theoretically feasible under the model, it is computationally infeasible.

5. **Many-To-Many Correspondence:**

The many-to-many correspondence existing between the plaintext and cipher text in the algorithm makes it impervious to the following attacks [3]:

1)a) Cipher-text only attack

Given :C1 = Ek(P1),C2=Ek(P2), . . . , Ci=Ek(Pi)

Deduce: Either P1, P2...Pi; K; or an algorithm to infer Pi+1 from Ci+1 = Ek(Pi+1)

1)b) Chosen Cipher-text attack

Given : C1, P =Dk(C1), C2,P2=Dk(C2), . . . ,

Ci, Pi=Dk(Ci)

Deduce: K [1].

A plaintext block P_1 uses a base B_1 to generate a cipher C_1. If the same plaintext block P_1 occurs again it will use a different base B_2, as the base is independent of plaintext pattern, and generates cipher C_2 as shown in fig. 8. Thus, a one-to-many relationship between the plaintext and the cipher is established. Because of this one-to-many relationship, both the cipher-text only attack and Chosen Cipher-text attack fails.

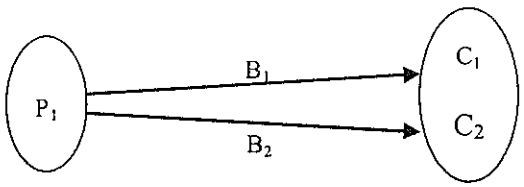


Fig.8: One to Many Mapping

2)a) Known plaintext attack:

Given: P1, C1= Ek(P1), P2, C2=Ek(P2), . . . , Pi,

Ci=Ek(Pi)

Deduce: Either K; or an algorithm to infer Pi+1

from Ci+1 =Ek(Pi+1)

2)b) Chosen plaintext attack:

Given: P1, C1 = Ek(P1), P2, C2=Ek(P2), . . . ,

Pi, Ci=Ek(Pi),

where the cryptanalyst gets to choose P1, P2,...,Pi

Deduce: Either K; or an algorithm to infer Pi+1 from Ci+1 = Ek(Pi+1)

A Cipher C_1 can be obtained as follows:

Cipher C_1 = status_vector$_i$ XOR $BaseX_m$ .....(1)

Cipher C_1 can also be obtained as:

Cipher C1 = status_vector$_j$ XOR $BaseX_n$ .....(2)

Two different plaintext blocks can thus produce the same cipher as shown in figure 9.

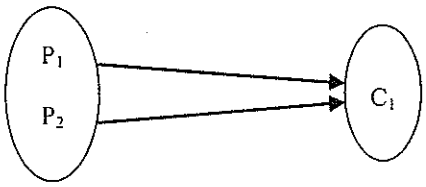


Fig.9: Many to One Mapping

The total number of base patterns that could be

obtained is $2^{64} P_{64}$.

$= 2^{64} \times (2^{64} - 1) \times (2^{64} - 2) \times .... \times (2^{64} - 64)$

$= 2^{64 \times 64}$ .....(3)

Because of the existence of large number of base patterns as shown in equation 3, the known plaintext and chosen plaintext attacks are practically infeasible.

6. **Test For Randomness – Auto-Correlation Test:**

Correlation [4] between the secret key and the output of the cryptosystem is the main source of information to the cryptanalyst. Given a cipher test sample: C_1, C_2 ... C_L, starting with t=1; the total number of occurrences $C_i = C_{i+t}$ for $1 <= i <= L-t$ is counted. This was repeated for

t=1, 2, 3... and a graph were plotted as shown in fig. 6. The graph shown in fig. 10 implies that the probability distribution of the cipher will be a nearly uniform distribution, thus forming a *digital envelope*.

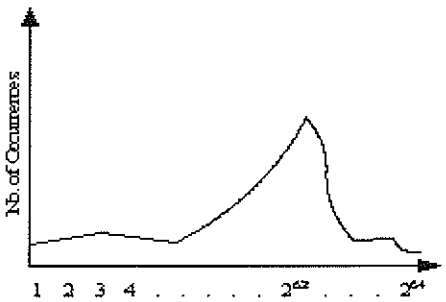


Fig.10: Experimental Graph

4. Illustration

4.1 Encryption

This section explains the working of the function encrypt and decrypt with an illustration using 8-bit plaintext. Let the plaintext block to be encrypted be:

Plaintext: 0 0 0 0 1 1 1 0

Let the base string, i.e., *basestr* obtained from the base[] array domain be:

basestr: 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1

Let the function framed from rand [Val] be

F[x]: $2x^2 + 7x + 190$

Index values are generated by substituting x= 1 to 8 in the above function.

199	212	229	250	275	304	337	374

Fig.11: Index Vector

The base vector is traversed and the bit values at the appropriate locations as given by the index vector are used to build the observed vector.

Observed vector: 0 0 1 0 1 1 0 0

The XOR operation between the input vector and the observed vector yields the status vector as shown in fig.12.

Input Vector: 0 0 0 0 1 1 1 0

Observed Vector: 0 0 1 0 1 1 0 0 **(XOR)**

Status Vector: 0 0 1 0 0 0 1 0

Fig.12: Obtaining the Status Vector

baseX is obtained by XORing two 8-bit parts of basestr as shown in Fig. 13.

basestr : 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1

basestr part1: 1 1 1 1 0 0 0 0

basestr part2: 0 0 0 0 1 1 1 1 **(XOR)**

BaseX : 1 1 1 1 1 1 1 1

Fig.13: Obtaining baseX

This status_vector is XORed with baseX to produce the Cipher text as shown in Fig. 14.

BaseX : 1 1 1 1 1 1 1 1

Status_vector: 0 0 1 0 0 0 1 0 **(XOR)**

Final Cipher : 1 1 0 1 1 1 0 1

Fig.14: Obtaining the final cipher text

4.2 Decryption:

The same function is framed from rand[val] at the receiver end.

F[x]: $2x^2 + 7x + 190$

Assigning the value of x = 1 to 8 to the function, the index vector in fig.15 is obtained.

199	212	229	250	275	304	337	374

Fig.15: Index Vector

The base string obtained, i.e., *basestr* is

basestr : 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1

Similar to the encryption module the value of baseX is obtained from basestr.

The status vector is obtained by XORing the cipher text and the baseX as shown below.

baseX	**:**	**1 1 1 1 1 1 1 1**
Final Cipher	**:**	**1 1 0 1 1 1 0 1 (XOR)**
Status_vector	**:**	**0 0 1 0 0 0 1 0**

Fig.16: Obtaining the final cipher text

By traversing the repeating base string passed to the function, the bit values from the positions given by the index vector is used to obtain the Observed vector.

Status vector	**:**	**0 0 1 0 0 0 1 0**
Observed Vector	**:**	**0 0 1 0 1 1 0 0 (XOR)**
Input String	**:**	**0 0 0 0 1 1 1 0**

Fig.17: Obtaining the Plain text

This procedure yields the decrypted 8-bit plain text as:

Plaintext: 0 0 0 0 1 1 1 0

5. Conclusion

A block cipher that possesses the desirable characteristics of a secure crypto algorithm has been proposed in this paper. Empirical analysis of the algorithm has shown that it is invulnerable to known-plaintext, chosen-plain-text and cipher-text only attacks. With the growth of VPN as a chief mode of message transfer for most of the business organizations, the proposed algorithm can be used to provide privacy and integrity of messages traveling across VPN. This paper also gives an illustration of the working of the proposed method . The strengths of the algorithm has also been discussed in this paper. The proposed method also achieves an input output bit conversion ratio of 1:1.

6. References

[1] Klavs Klavsen, "Securing remote users VPN access to your Company LAN", SANS Institute, July 29 , 2001.

[2] Pawel Golen, "Virtual Private Networking", www.WindowSecurity.com, July 22, 2004

[3] A. Menezes, P. van Oorschot and S.Vanstone, Handbook of Applied Cryptography, CRC Press, 2001.

[4] Bruce Schneier, Applied Cryptography, Second Edition, Singapore, John Wiley & Sons Inc., 2002.

[5] Sarah Granger, Unlocking the Secrets of Crypto: Cryptography, Encryption, and Cryptology Explained, www.securityfocus.com/infocus/1617, August 2002.

[6] Donald E. Knuth, Volume 2 - Seminumerical Algorithms, The Art of Computer Programming, Second Edition, Addison-Wesley, 2000.

[7] Promonia Secure Business Solutions, Secure Broker, http://www.promia.com/ products_and_tools / secure_broker/ Secure_Broker.htm, Accessed:12-Sept-2003.

[8] Joris Claessens, A Secure European System for Applications in a Multi-vendor Environment , https:/ /www.cosic.esat.kuleuven.ac.be/sesame/, Accessed-Sept-12-2003

[9] M Stevan Bellowin, Probable Plaintext Cryptanalysis of the IP Security Protocols, AT/T labs research., 1997.

[10] Microsoft, Windows 2000 kerberos authentication, www.microsoft.com,USA, 2000.

[11] Tom Davis, RSA encryption, www.geometer.org/ mathcirdes, October 2003.

Author's Biography :

Ms G Sudha Sadasivam, obtained her ME, Computer Science and Engineering from PSG College of Technology. She is pusuing her research work in the area of Distributed Object Computing. She is working as an Assistant Professor, Department of CSE, PSG College of Technology, Coimbatore. Her areas of interest include Distributed Systems, Compilers, Network Security and Object Computing. She has published 6 papers in National and International journals and 14 papers in National and International Conferences.