# An Embedding technique for encoding for secure and lossless transmission

*Candidate Name: Ravi shankar Raja Allam Mtech(CSe) Email id: ravisankar.raja@gmail.com*
*VISAKHA institute of engineering and technology*
*Guide: A Ajaya Kumar Mtech Cse(head of department)*

**Abstract:**We propose new asynchronous client(s)/server architecture on acknowledged protocols(TCP). The clients will be sending requests asynchronously to the multi threaded asynchronous server where secured padding decoder is available. Basic client service architectures are based on tired or clustered based architecture. The main disadvantage of this tired/clusters is clumsy and time consumption for real time deployment. A new embedding(padding) encoder/decoder algorithm used for lossless transmission. Once the server started the client(s) can concurrently transmits encoded requests to the server. All the clients requests will be processed at the server side and packets will be logged for the dedicated IP(trusted IPs).

*(**index terms**: asynchronous, padding)*

**Literature survey:**

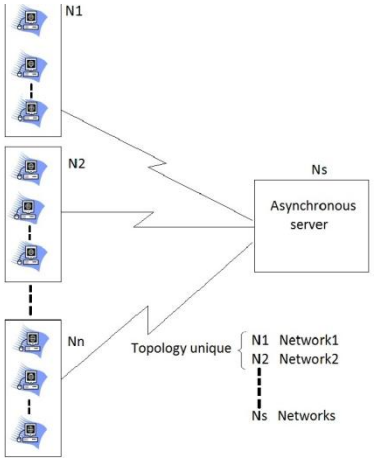
- Steps to design to protect any network (legacy , popular) are indicated as network security.
- If any network is securely reliable, it will be targeted a wide variety of attacks and for compromised nodes.
- To keep network more secured continuous monitoring on routers with credentials is most important.
- Transmission of the cross networks is most important aspect in the network security.
- Unique encoding techniques has to be updated continuously for better security for networks from the current traffic attacking trends.
- Vulnerability is the main aspect in the network security.

**Introduction:**

Our work is mainly on legacy and most popular topologies. The reason most of the sectors will be still using the old networks and the reason is migration to the new topologies is time and cost effective. So this system is open to adopt old network topologies without changing the regular existing network. Our system can be open for clients can be deployed in cross level networks.

The encoded transmission will be logged for dedicated and trusted peers(clients) for further migration of the peers as important nodes. Once the clients starts the communications all the requests will be processed asynchronously with our system for better time improvements.

**Architecture:**




**Algorithm:**

***Padding Encoding:***

Padding technique is a group of similar binary-to-text encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation.

Padding encoding schemes are commonly used when there is a need to encode binary data that needs to be stored and transferred over media that are designed to deal with textual data. This is to ensure that the data remain intact without modification during transport. Padding is commonly used in a Padding encoding takes the original binary data and operates on it by dividing it into tokens of three bytes. A byte consists of eight bits, so Padding takes 24bits in total. These 3 bytes are then converted into four

printable characters from the ASCII standard.

The algorithm's name Padding comes from the use of these 64 ASCII characters. The ASCII characters used for Padding are the numbers 0-9, the alphabets 26 lowercase and 26 uppercase characters plus two extra characters '+' and '/'.

The first step is to take the three bytes (24bit) of binary data and split it into four numbers of six bits. Because the ASCII standard defines the use of seven bits, Padding only uses 6 bits (corresponding to 2^6 = 64 characters) to ensure the encoded data is printable and none of the special characters available in ASCII are used.

The ASCII conversion of 3-byte, 24-bit groups is repeated until the whole sequence of original data bytes is encoded. To ensure the encoded data can be properly printed and does not exceed the limit.

When the number of bytes to encode is not divisible by 3 (that is, if there are only one or two bytes of input for the last 24-bit block), then the following action is performed: Add extra bytes with value zero so there are three bytes, and perform the conversion to Padding. If there was only one significant input byte, only the first two Padding digits are picked (12 bits), and if there were two significant input bytes, the first three Padding digits are picked (18 bits). '=' characters might be added to make the last block contain four Padding characters.

***Example:***

| | | | | |
|---|---|---|---|---|
| ***Text content*** | ***M*** | ***a*** | ***n*** | |
| ***ASCII*** | ***77*** | ***97*** | ***110*** | |
| ***Bit pattern*** | ***0 1 0 0 1 1 0 1 0 1 1 0 0 0 0 1 0 1 1 0 1 1 1 0*** | | | |
| ***Index*** | ***19*** | ***22*** | ***5*** | ***46*** |
| ***padding-encoded*** | ***T*** | ***W*** | ***F*** | ***u*** |

***Padding:***

The '==' sequence indicates that the last group contained only 1 byte, and '=' indicates that it contained 2 bytes.

***Example1:***

Input:
*any carnal plea**sure.***
Output:
YW55IGNhcm5hbCBwbGVhc3VyZS4=

***Exapmple2:***

Input:
*any carnal plea**sure***
Output:
YW55IGNhcm5hbCBwbGVh**c3VyZQ==**

***Algorithm3:*** Padding ***(encoding) algorithm:***

Input ⟶ raw string

Output ⟶ Padding encoded format

***Step1:*****initialization**

$$\int \text{ALPHABET} = \sum_0^{25} \text{CAPS}(65-90) + \sum_0^{25} \text{SMALL}(97-122) + \sum_0^{9} \text{NUMBERS}(48-57) + \sum_0^{2} \text{SPC}(43,67)$$

// all "ALPHABET" CONTAINS ASCII values for capital letters, small letters, single digit numbers, '+' and '/' characters.

***Step2:***

***Functionality:***

To convert alphabets to ASCII codes

Input ⟵ all available characters

Output ← all equivalent ASCII values

n ← 0

B0 ← 0

B1 ← 0

B2 ← 0

$\sum_0^n$ buff[] ← 0

$\sum_0^n$ ar ← 0

**i** ← 0

Iteration ← 0

Loop statement:

Count=0

For each C in ALPHABET

toInt (count) = TOINT (ALPHABET (i))

Count++

End loop

Size=SIZE (buff)

Iteration ← (((size+2)/3)*4)

Do while n in iteration

B0 ← buff

B1 ← (i <size)? buff++: 0

B2 ← (i < size)? buff++: 0

***Masking***

Mask=0X3F

ar = ALPHABET [(b0>>2) & mask]

ar= ALPHABET [(b0<<4) | ((b1&0XFF)>>4)) & mask]

ar= ALPHABET [(b1<<2) | ((b2&0XFF)>>6)) & mask]

ar = ALPHABET [b2 & mask]

End while

**Padding:**

If (size % 3=1)

ar ← "="

ar ← "="

Else if (size % 3=2)

ar ← "="

else

size %3=0

End if

***Decoding:***

When decoding Padding text, 4 characters are typically converted back to 3 bytes. The only exceptions are when padding characters exist. A single '=' indicates that the 4 characters will decode to only 2 bytes, while 2 '='s indicates that the 4 characters will decode to only a single byte.

***Example:***

**Input:**
YW55IGNhcm5hbCBwbGVh**cw==**
Block with 2 '='s decodes to 1 character:
**Output:**
*any carnal plea**s***
Padding ***(decoding) algorithm:***

Input ← string

Output ← $\sum_0^n$ buff(bytes)

***Initialization:***

Buff ← 0

S ← string decode

N ← length of string

Mask ← 0XFF

If s [0] = '=='

Delta =2

Else if s [0] = '='

Delta =1

Else

Delta=0

End if

Loop

For I ← 0 step by 4 of in n

C0=Convert to Int [CharAt (i) in S]

C1=Convert to Int [CharAt (i+1) in S]

$buffer_i$ ← (c0<<2) | (c1>>4) & mask

C2 ← Convert to Int [CharAt (i+2) in S]

$buffer_{i+1}$ ← (c1<<4)|(c2>>2)&mask

C3=convert toInt [i+3]

$buffer_{i++}$ = (c2<<6)! c3&mask

End loop

**Protocols Descriptions:**

The above two algorithms describe the padding technique to get the printable ASCII characters. The whole procedure is all about the encoding and decoding patterns. This is necessarily done to ensure safe arrival of data in transport without any loss of packets. The encoding is done by getting an input string and converting it to its ASCII value so that the binary data is then appended and forms into a complete string. This string is then divided into 3 bytes, 24 bits. To these 24 bits we check for the required bit length, so that if at all there is any ambiguity in the length we can adjust it by appending zeros thus forming into a 24bit length string. We then divide this new string of bits into 4 groups; each group has 6bit length. Thus assuring it to be converted as printable character which forms the output for encoding. The decoding procedure is all about the reverse to the above procedure. We convert the encoded string into binary form and grouping them as 8 bit length each and to get the original string as the final output.

**Conclusion:**

Thus we conclude that in TCP transmission the clients can send packets of data without any losses by using this padding technique. This technique ensures the safeguarding of data in transport whether the servers or clients are asynchronous. It also ensures the data security by converting the original given string into printable characters which do not use any special characters except '+' and '/'.

**References:**

*[AD01a] ALLIEZ P., DESBRUN M.: Progressive encoding for lossless transmission of triangle meshes. In ACM SIGGRAPH (2001), pp. 198–205.*

*[AD01b] ALLIEZ P., DESBRUN M.: Valence-driven connectivity encoding for 3D meshes. In EUROGRAPHICS (2001), pp. 480–489.*

*[AFSR03] ATTENE M., FALCIDIENO B., SPAGNUOLO M., ROSSIGNAC J.: Swingwrapper: Retiling triangle meshes for better edgebreaker compression. ACM Transactions on Graphics 22, 4 (2003), 982–996.*

*[AG03] ALLIEZ P., GOTSMAN C.: Recent advances in compression of 3d meshes. In Proc. of the Symp. on Multiresolution in Geometric Modeling (Sep 2003).*

*[BPZ99a] BAJAJ C., PASCUCCI V., ZHUANG G.: Progressive compression and transmission of arbitrary triangular meshes. In IEEE Visualization (1999), pp. 307–316.*

*[BPZ99b] BAJAJ C. L., PASCUCCI V., ZHUANG G.: Single resolution compression of arbitrary triangular meshes with properties. Computational Geometry: Theory and Applications 14 (1999), 167–186.*

*[COLR99] COHEN-OR D., LEVIN D., REMEZ O.: Progressive compression of arbitrary triangular meshes. In IEEE Visualization (1999), pp. 67–72.*

*[CR04] COORS V., ROSSIGNAC J.: Delphi: geometry-based connectivity prediction in triangle mesh compression. The Visual Computer 20, 8-9 (2004), 507–520.*

*[CRS98] CIGNONI P., ROCCHINI C., SCOPIGNO R.: Metro: Measuring error on simplified surfaces. Comp. Graph. Forum 17, 2 (1998), 167–174.*

*[DG00] DEVILLERS O., GANDOIN P.: Geometric compression for interactive transmission. In IEEE Visualization (2000), pp. 319–326.*

*[GD02] GANDOIN P. M., DEVILLERS O.: Progressive lossless compression of arbitrary simplicial complexes. ACM Trans. Graphics 21, 3 (2002), 372–379.*

*[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. In ACM SIGGRAPH (2002), pp. 355–361.*

*[GGK02] GOTSMAN C., GUMHOLD S., KOBBELT L.: Simplification and compression of 3D meshes. In Tutorials on Multiresolution in Geometric Modelling (2002).*

*[GP05] GABRIEL PEYRÉ S. M.: Surface compression with geometric bandelets. In ACM SIGGRAPH (2005), pp. 601–608. [GS98] GUMHOLD S., STRASSER W.: Real time compression of triangle mesh connectivity. In ACM SIGGRAPH (1998), pp. 133–140.*

*[GWH01] GARLAND M., WILLMOTT A., HECKBERT P. S.: Hierarchical face clustering on polygonal surfaces. In SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics (2001), pp. 58. [Hop96] HOPPE H.: Progressive meshes. In ACM SIGGRAPH (1996), pp. 99–108.*

*[ILS04] ISENBURG M., LINDSTROM P., SNOEYINK J.: Lossless compression of floating-point geometry. In Proceedings of CAD'3D (2004), pp. 1–4.*

*[ILS05] ISENBURG M., LINDSTROM P., SNOEYINK J.: Lossless compression of predicted floating-point geometry. Computer-Aided Design 37, 8 (2005), 869–877.*