Docket No. MRTN01P

PROVISIONAL APPLICATION FOR PATENT

ANTI-SPAM EMAILING SYSTEM AND METHOD

INVENTORS:
Stephen Martin

Anti-Spam Emailing System and Method

by Stephen Martin

BACKGROUND OF THE INVENTION

[0001] Conventional anti-Spam techniques include content and IP based approaches. Spam filtering based on email content is known to often result in missing valid communications. Spam filtering based on blocking IP blocks sections of the internet's address space thereby effectively creating zones of information quarantine. FIG. 1 illustrates, by way of example, how the current RFC822 delivery mechanism works.

[0002] In view of the foregoing, there is a need for improved techniques for blocking Spam, while significantly reducing the risk of losing valid emails.

SUMMARY OF THE INVENTION

[0003] A variety of improved anti-Spam techniques are described. A goal of the present invention is to prevent internet users from receiving un-solicited emails, also referred to as Spam or ube.

[0004] Unlike conventional approaches, in one aspect of the present invention, the email content is not scanned or filtered. Moreover, communications are not blocked just because they happen to be coming from a specific Internet Protocol (IP) address. The present invention also renders pointless the current practice of scanning of internet world wide web content and internet news group content for email addresses to compile mailing lists that are often maliciously used for mass distribution Spamming. The present invention further substantially reduces, if not eliminates, the problem of receiving unwanted emails. Known anti-Spam techniques also suffer from falsely identifying, or flagging, valid email as Spam, this problem is often referred to as "false-positives." The present invention substantially reduces, if not eliminates, the false-positive problem.

[0005] In some embodiments of the present inventions, it is possible to reduce, by up to 96-100%, the amount of un-solicited email (Spam) that the average internet user receives. By blocking Spam mail, it is expected that there may be a reduction in the overall data traffic over many of the worlds email servers.

[0006] Another aspect of the present invention is that it can be 100% backward compatible with practically all existing mail servers and email clients.

[0007] Other features, advantages, and object of the present invention will become more apparent and be more readily understood from the following detailed description, which should be read in conjunction with the accompanying drawings.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0008] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements. Unless otherwise indicated illustrations in the figures are not necessarily drawn to scale. The present invention is best understood by reference to the detailed figures and description set forth herein.

[0009] Embodiments of the invention are discussed below with reference to the Figures. However, those skilled in the art will readily appreciate that the detailed description given herein with respect to these figures is for explanatory purposes as the invention extends beyond these limited embodiments.

[0010] Certain technical terminology used herein shall be set forth as follows:

[0011] Spam: Un-solicited, unwanted and often offensive email communications.

[0012] UCE: Un-solicited, unwanted and often offensive email communications.

[0013] UBE: Un-solicited, unwanted and often offensive email communications.

[0014] IP Address: Every machine on the Internet has a unique identifying number, called an IP Address.

[0015] ESMTP/SMTP: Simple Mail Transfer Protocol, a protocol for sending e-mail messages between servers.

[0016] “Calico’ing”: The act of extending an email address in an RFC compliant manner.

[0017] ”Calico’ed’ : An email address that has been extended using the present invention.

[0018] An enhanced Open Email Address Mechanism according to an embodiment of the present invention is illustrated by way of example in FIG. 2.

also known as “Calico’ing” utilizing a “Calico’ed” email address.

[0019] The present invention is capable of operating under a conventional computer system environment. Certain embodiments of the present invention are relatively operating system independent. For example, the present invention may be successfully implemented on Microsoft Windows and Unix based operating systems. Moreover, the present invention operates within both a home Internet user and commercial environment.

[0020] By way of example, and not limitation, the present invention can exist either as a component with server software (usually an SMTP server) or as a configuration change to carry out the prescribed behavior. The configuration change capability enables the present invention to operate without the need to modify any existing server software.

[0021] To effectively prevent unwanted email reaching a mailbox, one aspect of the present invention includes in the email address both the current user identification information and additional authentication information.

[0022] One embodiment of the present invention is an extension to the existing RFC822 email format of someuser@somedomain.com. This extended format of the present embodiment is still RFC822 compliant, hence, these extended addresses can travel through existing mail servers and can be processed for sending by existing email clients, including, for example, applications like Outlook, Netscape and Eudora.

[0023] Another aspect of the present invention is to not think of email addresses as single entities that will allow delivery of email regardless of content of solicitation. Instead, email addresses are viewed in terms of being open or closed. On a Calico'ed

system, according to an embodiment of the present invention, email addresses are divided into two modes, specified as follows:

closed: **someuser@somedomain.com**

and

open: ***auth_code*#someuser@somedomain.com**

[0024] Email under the present embodiment will not be delivered on a Calico'ed system unless the sender has obtained a predefined authorization code. The authorization code may be obtained in any number of ways. In some embodiments the authorization code will either have been verbally communicated to the sender by the recipient, or it will appear as text next to the recipient's email address (for example, displayed on a web page). In alternative embodiments, the recipient may obtain a sender's authorization code when the sender sends at least one email to the recipient using the extended open format, which is RFC822 compliant. As illustrated in FIG. 2, authentication of the sender's authorization code is performed by what is herein referred to a "Calico Engine." The "Calico Engine" that authenticates the extended email address reside in any convenient location in the email transport path. In many implementations, the "Calico Engine" could reside on a server. Those in the art will appreciate that the "Calico Engine" according to the present invention may be implemented in a multiplicity of known ways, including, by way of example and not limitation, specially written email server software or by configuration changes to existing email server software.

[0025] Those skilled in the art will know how to implement the principles of the present invention using known programming languages and operating systems. For example, in a Microsoft Windows™ operating system based mail server, the above delivery mechanism may be implemented by the following Microsoft ™ Visual Basic 6.0 program code segment that implements the Calico mechanism as also illustrated by way of example in the flow chart of FIG. 3.

<CODE>

```
Private Sub ESMTPServer_OnValidateRCPTTO(ID As String, User As String, Domain As String, IIIToken As String, Valid As Boolean)

    Dim sAccount$, lFoundIndex&, sParam$
```

```
    Valid = False

    sAccount = User & "@" & Domain

    lFoundIndex = IndexSearch(gMailboxes.Index, sAccount)
    If lFoundIndex >= 0 Then
       '-- Check III Key
       If gMailboxes.Data(lFoundIndex).UseKeyAuthenticate Then
          sParam = modIIIKey.ProcDecrypt(IIIToken,
gMailboxes.Data(lFoundIndex).PublicKey)
          If StrComp(sParam, Domain, vbTextCompare) <> 0 Then
             LogEvent ID, "WARNING Invalid IIIToken: " & IIIToken & " for " & sAccount
          Else
             LogEvent ID, "Valid IIIToken: " & IIIToken & " for " & sAccount
             Valid = True
          End If
       Else
          Valid = True
       End If
    Else
       LogEvent ID, "WARNING: Invalid recipient " & sAccount
    End If
End Sub

Public Function IndexSearch(IndexList() As Z_INDEX, FindText As String, Optional BinaryCompare As Boolean =
False, Optional ReturnInsertionPoint As Boolean = False) As Long

    Dim lIndex&, lMax&, lMin&, lMid&, iComp%, iMethod%

    lIndex = -1

    On Error Resume Next
    lMin = LBound(IndexList)
    lMax = UBound(IndexList)
    If Err Then
       Err.Clear
       IndexSearch = -1
       Exit Function
    Else

       If BinaryCompare Then
          iMethod% = vbBinaryCompare
       Else
          iMethod% = vbTextCompare
       End If

       '-- Do a Binary Search

       If lMax > lMin Then

          Do
             If lMax < lMin Then
                Exit Do
             End If

             lMid = lMin + ((lMax - lMin) \ 2)

             iComp = StrComp(IndexList(lMid).KeyValue, FindText, iMethod%)
```

```
                Select Case iComp
                    Case 0
                        ' Item found
                        IndexSearch = IndexList(lMid).ArrayIndex
                        Exit Function

                    Case Is > 0
                        ' Item is in lower half
                        lMax = lMid - 1
                        If lMin = lMax Then Exit Do

                    Case Is < 0
                        ' Item is in upper half
                        lMin = lMid + 1
                        If lMin = lMax Then Exit Do
                End Select
            Loop
        End If

        '-- Check to See if this is the one
        If StrComp(IndexList(lMin).KeyValue, FindText, iMethod%) = 0 Then
            ' Item found
            lIndex = IndexList(lMin).ArrayIndex
        Else
            If ReturnInsertionPoint Then
                ' Item not found, but return position to insert
                lIndex = lMid - (iComp < 0)
                lIndex = IndexList(lMid).ArrayIndex
            Else
                lIndex = -1
            End If
        End If

    End If

    IndexSearch = lIndex

End Function
Public Function IndexSort(IndexList() As Z_INDEX, lLBound As Long, lUBound As Long, Optional
CompareMethod As VbCompareMethod = vbTextCompare) As Integer

    Dim strTemp As String
    Dim IndexBuffer As Z_INDEX
    Dim lngCurLow As Long
    Dim lngCurHigh As Long
    Dim lngCurMidpoint As Long
    Dim iCompareOption As Integer


    ' Start current low and high at actual low/high
    lngCurLow = lLBound
    lngCurHigh = lUBound

    If lUBound < (lLBound + 1) Then
        If (lUBound = lLBound) And lLBound = 0 Then
            IndexSort = 1
        Else
            IndexSort = lUBound
        End If

        Exit Function
    End If

    On Error Resume Next
    If UBound(IndexList) < lUBound Then Exit Function

      '--Find the approx midpoint of the array
```

```
        lngCurMidpoint = (lLBound + lUBound) \ 2

        '--Pick as a starting point (we are making
        '     an assumption that the data *might* be
        '     in semi-sorted order already!

         strTemp = IndexList(lngCurMidpoint).KeyValue

         Do While (lngCurLow <= lngCurHigh)

            Do While StrComp(IndexList(lngCurLow).KeyValue, strTemp, CompareMethod) < 0
               lngCurLow = lngCurLow + 1
               If lngCurLow = lUBound Then Exit Do
            Loop

            Do While StrComp(strTemp, IndexList(lngCurHigh).KeyValue, CompareMethod) < 0
               lngCurHigh = lngCurHigh - 1
               If lngCurHigh = lLBound Then Exit Do
            Loop


            If (lngCurLow <= lngCurHigh) Then
              '--if low is <= high then swap
              IndexBuffer = IndexList(lngCurLow)
              IndexList(lngCurLow) = IndexList(lngCurHigh)
              IndexList(lngCurHigh) = IndexBuffer

              lngCurLow = lngCurLow + 1 ' CurLow++
              lngCurHigh = lngCurHigh - 1 ' CurLow--
            End If

         Loop

         If lLBound < lngCurHigh Then
            '-- Recurse if necessary
            IndexSort IndexList(), lLBound, lngCurHigh, CompareMethod
         End If

         If lngCurLow < lUBound Then
            '-- Recurse if necessary
            IndexSort IndexList(), lngCurLow, lUBound, CompareMethod
         End If

      IndexSort = lUBound - lLBound + 1
   End Function

   Public Function ProcDecrypt(Source As String, Key As String) As String

      Dim s$, b%, k1$, k2$, k3$, k_2$, o, r1$, r2$, w2$
      Dim i%, l&, iLen%, lLen&, iChar%, sTemp$

      s = Source
      k1 = Key

   '   $_s = reverse($_s);
      lLen = Len(s)
      sTemp = ""
      For l = lLen To 1 Step -1
         sTemp = sTemp & Mid$(s, l, 1)
      Next
      s = sTemp

   '  $_s =~ s/\./a/g;
   "    s = Replace(s, ".", "a", , , vbBinaryCompare)
   '  $_s =~ s/-/b/g;
   "    s = Replace(s, "-", "b", , , vbBinaryCompare)
```

```
'    $_s =~ s/\+/c/g;
"     s = Replace(s, "+", "c", , , vbBinaryCompare)
'    $_s =~ s/\!/d/g;
"     s = Replace(s, "!", "d", , , vbBinaryCompare)
'    $_s =~ s/\=/e/g;
"     s = Replace(s, "=", "e", , , vbBinaryCompare)
'    $_s =~ s/\^/f/g;
"     s = Replace(s, "^", "f", , , vbBinaryCompare)

'   $_b=ord(substr($_k1,0,1));
    b% = Asc(Mid$(k1, 1, 1))
'   $_k2=$_b ** 7;
    k2 = "" & (CDbl(b) ^ 7#)
'   @_k_2=split(/"/,$_k2);
'   $_k2=join '',@_k_2;

'   $_k3=reverse($_k1);
'   $_k3 =~ tr /[a-m][n-z]/[n-z][a-m]/;
'   $_k3 =~ tr /[A-M][N-Z]/[N-Z][A-M]/;

    '-- Reverse and transliterate key
    iLen = Len(k1)
    For i% = iLen To 1 Step -1
        iChar = Asc(Mid$(k1, i, 1))
        If iChar >= 97 And iChar <= 109 Then
            '-- [a-m] to [n-z]
            iChar = iChar + 13
        ElseIf iChar >= 110 And iChar <= 122 Then
            '-- [n-z] to [a-m]
            iChar = iChar - 13
        ElseIf iChar >= 65 And iChar <= 77 Then
            '-- [A-M] to [N-Z]
            iChar = iChar + 13
        ElseIf iChar >= 78 And iChar <= 90 Then
            '-- [N-Z] to [A-M]
            iChar = iChar - 13
        End If

        k3 = k3 & Chr$(iChar)
    Next

'    for($_d=0;$_d < length($_s);$_d=$_d+2) {
'       $_w1[$_d]=chr(hex(substr($_s,$_d,2)));
'                                       }
'    $_w2=join ('',@_w1);
    lLen = Len(s)
    For l = 1 To lLen Step 2
        w2 = w2 & Chr$(Int("&h" & Mid$(s, l, 2)))
    Next

'   $_r1 = _xor($_w2,"$_k3");
    r1 = ProcXor(w2, k3)
'   $_r2 = _xor($_r1,"$_k2");
    r2 = ProcXor(r1, k2)
'   $_o =  _xor($_r2,"$_k1");
    o = ProcXor(r2, k1)

'   return $_o;
    ProcDecrypt = o

End Function
Public Sub LogEvent(ID As String, Description As String)

    Dim i%, Sid$, iCount%, sEvent$, sArray$(), fNum%, sFile$

    If Trim(Description) = "" Then Exit Sub
```

```
    If Trim(ID) <> "" Then
       Sid = " " & Trim(ID)
    Else
       Sid = ""
    End If

    sEvent = Replace(Description, vbCr, "")

    '-- Ensure Log Path
    On Error Resume Next
    MkDir App.Path & "\Logs"
    On Error GoTo 0

    fNum = FreeFile
    sFile = App.Path + "\Logs\" + Format(Now, "yyyy mmm dd.log")
    Open sFile For Append As #fNum

    If Left$(Description, 1) <> vbTab Then
       sArray = Split(sEvent, vbLf)

       For i% = 0 To UBound(sArray)
          If i% = 0 Then
             sEvent = Format(Now, "h:nn:ss am/pm") & Sid & vbTab & sArray(i)
          Else
             sEvent = vbTab & vbTab & sArray(i)
          End If

          Print #fNum, sEvent
          lstLog.AddItem sEvent
       Next
    Else
       Print #fNum, sEvent
       lstLog.AddItem vbTab & sEvent
    End If

    Close fNum

    iCount = lstLog.ListCount
    If iCount > 250 Then
       '-- Remove oldest items
       For i% = iCount To 250 Step -1
          lstLog.RemoveItem 0
       Next
    End If

    lstLog.ListIndex = lstLog.NewIndex
End Sub
```

</CODE>

[0026] The above code segment first determines at Step 305 if the user account specified in the email address is serviced by the Mail server. If not, an error is logged and the mail is NOT delivered. If the user account is serviced, as determined at Step 310, then the authorization phrase supplied in the extended email address is decrypted, at Step 315, and compared, at Step 320, to the authorization phrase supplied by the user (account holder). If they are not equal either due to an incorrect phrase or non-existent phrase (for example, possibly because the email is an un-solicited bulk email) then the email is rejected. If the phrases match then email delivery occurs at Step 325.

[0027] In some embodiments, Windows or other GUI based operating systems would initially use an Open Email Address Generator to construct the extended open (Calico'ed) email address. FIG. 4 shows an exemplary flowchart of a MS Windows based embodiment Open Email Address Generator (OEAG) program that takes inputs via a GUI form 405 and generates an extended "Calico'ed" email address; i.e., an extended open email address. By way of further example of alternative implementation methods, the following Microsoft™ Visual Basic 6.0 code segment is intended to implement the embodiment of FIG. 4:

```
<CODE>
Private Sub Command1_Click()
On Error GoTo ERR_Command1_Click  '~VB Crash Shield~
  Dim strUser As String
  Dim strDomain As String
  Dim strString As String
  Dim arrClosed() As String
  Dim strAuthPhrase As String
  Dim strOpenAddr As String


  If Me.Text1.Text <> "" Then
    arrClosed = Split(Me.Text1.Text, "@")

    strUser = arrClosed(0)
    strDomain = arrClosed(1)

    If Me.Text2.Text <> "" Then

    strAuthPhrase = Me.Text2.Text

    ' We haveboth closed address and AUTHPHRASE Present

     strOpenAddr = ProcEncrypt(strDomain, strAuthPhrase)

    strOpenAddr = strOpenAddr & "#" & LTrim(RTrim(Me.Text1.Text))
    Me.Text3.Text = strOpenAddr

    End If

  End If

Exit Sub  '~VB Crash Shield~
ERR_Command1_Click:  '~VB Crash Shield~
  Resume Next  '~VB Crash Shield~
End Sub

Public Function ProcEncrypt(Source As String, Key As String) As String
```

```
    Dim s$, b%, k1$, k2$, k3$, k_2$, e$, r1$, r2$, r3$
    Dim i%, l&, iLen%, lLen&, iChar%, stemp$

    s = Source
    k1 = Key

'   $_b=ord(substr($_k1,0,1));
    b% = Asc(Mid$(k1, 1, 1))
'   $_k2=$_b ** 7;
    k2 = "" & (CDbl(b) ^ 7#)

'   @_k_2=split(/"/,$_k2);
'   $_k2=join '',@_k_2;

'   $_k3=reverse($_k1);
'   $_k3 =~ tr /[a-m][n-z]/[n-z][a-m]/;
'   $_k3 =~ tr /[A-M][N-Z]/[N-Z][A-M]/;

    '-- Reverse and transliterate key
    iLen = Len(k1)
    For i% = iLen To 1 Step -1
        iChar = Asc(Mid$(k1, i, 1))
        If iChar >= 97 And iChar <= 109 Then
            '-- [a-m] to [n-z]
            iChar = iChar + 13
        ElseIf iChar >= 110 And iChar <= 122 Then
            '-- [n-z] to [a-m]
            iChar = iChar - 13
        ElseIf iChar >= 65 And iChar <= 77 Then
            '-- [A-M] to [N-Z]
            iChar = iChar + 13
        ElseIf iChar >= 78 And iChar <= 90 Then
            '-- [N-Z] to [A-M]
            iChar = iChar - 13
        End If

        k3 = k3 & Chr$(iChar)
    Next

'   $_r1 = _xor($_s,"$_k1");
    r1 = ProcXor(s, k1)
'   $_r2 = _xor($_r1,"$_k2");
    r2 = ProcXor(r1, k2)
'   $_r3 = _xor($_r2,"$_k3");
    r3 = ProcXor(r2, k3)

'   for($_d=0;$_d < length($_r3);$_d++) {
'       $_t[$_d]=sprintf("%02x",ord(substr($_r3,$_d,1)));
'                               }
'        $_e = join '',@_t;
    lLen = Len(r3)
    For l = 1 To lLen
        e = e & LCase(Right$("0" & Hex$(Asc(Mid$(r3, l, 1))), 2))
    Next

'   $_e =~ s/a/\./g;
"     e = Replace(e, "a", ".", , , vbBinaryCompare)
'   $_e =~ s/b/-/g;
"     e = Replace(e, "b", "-", , , vbBinaryCompare)
'   $_e =~ s/c/\+/g;
"     e = Replace(e, "c", "+", , , vbBinaryCompare)
'   $_e =~ s/d/\!/g;
"     e = Replace(e, "d", "!", , , vbBinaryCompare)
'   $_e =~ s/e/\=/g;
"     e = Replace(e, "e", "=", , , vbBinaryCompare)
'   $_e =~ s/f/\^/g;
"     e = Replace(e, "f", "^", , , vbBinaryCompare)
```

```
'   return reverse($_e);
    lLen = Len(e)
    stemp = ""
    For l = lLen To 1 Step -1
       stemp = stemp & Mid$(e, l, 1)
    Next

    ProcEncrypt = stemp

End Function
```

</CODE>

[0028] As an example of the email server configuration aspect of the present invention as directed to the Unix™ based Operating Systems using the Open Source mail server software "SendMail" (distributed by www.sendmail.org), the following configuration modifications allow SendMail to operate in a Calico'ed mode according to the embodiment of FIG. 3:

<CODE>

```
# Access list database (for spam stomping)
Kaccess hash -T<TMPF> -o /etc/mail/access

# Mailer table (overriding domains)
Kmailertable hash -o /etc/mail/mailertable

# Virtual user table (maps incoming users)
Kvirtuser hash -o /etc/mail/virtusertable
```

</CODE>

<CODE>

R<CLOSEDADDR> $* $#error $@ 5.2.1 $: 550 This recipient has a closed email address you must use an Open Email Address Generator to affect a delivery

</CODE>

<CODE>

```
# Billing email
to:billing@calicomail.com       CLOSEDADDR
to:761355f0f4254f052f14d414f6d3#billing@calicomail.com OK
```

</CODE>

[0029] The ability of the present invention to be readily adapted to a Unix™ based mail server by way of relatively simple software configured significantly reduces, if not eliminates, costly code re-writes of its programming code, which advantageously are not required for the mail server to operate in Extended "Calico'ed" Email Address mode

according to the present invention. Moreover, this adaptation capability significantly enhances the RFC822 compatibility of this embodiment of the present invention.

[0030] Unix™ based systems can then generate the correct encrypted authentication sequence by using a multiplicity of know approaches including script languages such as Perl (www.perl.org).

[0031] It should be appreciated, however, that unlike the form based program exemplified in FIG. 4, the results are written in a Unix™ based mail server's configuration files. In one embodiment of the present invention, the configuration files when used in conjunction with the configuration modification lines list just above, allow the open source SendMail mail server to process extended open email requests. Those skilled in the art will readily recognize a multiplicity of equivalent or alternative approaches.

[0032] What follows is an exemplary software code segment for a command line equivalent of the Windows™ based Open Email Address Generator (OEAG) utility program implemented Perl code:

```
<CODE>
# Open Live files for append

open(F1,">>$livealiasfile") || die "Cannot open $livealiasfile for appending $!\n";
open(F2,">>$liveaccess")  || die "Cannot open $liveaccess for appending $!\n";
open(F3,">>$livevirt")      || die "Cannot open $livevirt for appending $!\n";
open(F4,">>$ftpf")          || die "Cannot open $ftpf for appending $!\n";

while( $data = $sth->fetchrow_hashref())
{
 # Rebuild all mail files within this loop

 $mailbox=$data->{mb};
 $username=$mailbox;
 $mailbox = "cm" . $mailbox;

 $auth=$data->{auth};

 $hash = $y->crypt( $domain, $auth );

 print F1 "$hash#$username\@$domain: $mailbox\n";

 print F2 "to:$hash#$username\@$domain OK\n", "to:$username\@$domain CLOSEDADDR\n";

 print F3 "$hash#$username\@$domain\t$mailbox\n";

 print F4 $mailbox,"\n";

}
```

```
$rv = $sth->finish();

# Close files

close(F4);
close(F3);
close(F2);
close(F1);
</CODE>
```

[0033] Those skilled in the art will further appreciate that extended open email address's may also be readily generated from a users form input by way of an XML web service.

[0034] It should be appreciated that any suitable authorization key generation method may be used. In one embodiment of the encryption authorization key generator the encrypted authorization key is generated using a relatively simple encryption routine. Hence, the present encryption mechanism may be either strong or weak. Alternative embodiments may instead use strong 128 bit encryption instead, whereby strong 128 bit encryption is used to generate authorization phrase. However, depending on the application more simplistic approaches may be more suitable especially when the resultant cipher is required to be both completely ASCII represent able and not subject to any possible export restrictions. In yet other alternative embodiments of the present invention, the encryption may be derived by any regular arithmetic algorithm, "Enigma" style rotor algorithm, logarithmic algorithm, or fractal algorithm.

[0035] Those skilled in the art will readily recognize that Open Email Address Generation approach (e.g., a Calico'ed email address) according to the present invention and PGP (Pretty Good Protection by Verisign, Inc., for example) email, or any similar digital certificate exchange mechanism, are mutually exclusive. Email messages in the present invention are not necessarily encrypted; that is, encryption is merely one mechanism, or embodiment, that could be employed to generate the pre-appended authorization phrase. Furthermore, it should also be noted that because this mechanism is RFC822 compliant it does not hinder any security minded user from employing a PGP type mechanism to encrypt their message content.

[0036] The encryption algorithm employed by the present invention may be implemented in a multiplicity of known ways such that only ASCII (printable) characters are returned. For example, one suitable algorithm used was developed by Stephen Martin

in 1998 and initially released as a Perl module for Perl programmers. The algorithm ensures that only ASCII (printable) characters are returned. This makes it a suitable candidate for use in the present invention at least because non-printable characters would not be desirable. The algorithm is somewhat obscure, however it is also employed in other P3P Tools Inc, software such as Perl source code encryptors, Microsoft Excel™ add-in's and several other in-house utilities. The encryption is referred to as IIIKey or CEE depending on its usage. What follows is an exemplary code segment implementing the IIIKey encryption algorithm.

<CODE>

```
sub crypt {
   shift;
   my ($_s) = @_;
   shift;
   my ($_k1) = @_;
   my $_b    = "";
   my $_d    = 0;
   my $_k2   = "";
   my $_k3   = "";
   my @_k_2  = ();
   my @_t    = ();
   my $_e    = "";
   my $_r1   = "";
   my $_r2   = "";
   my $_r3   = "";

   $_b   = ord( substr( $_k1, 0, 1 ) );
   $_k2  = $_b**7;
   @_k_2 = split( /''/, $_k2 );
   $_k2  = join '', @_k_2;
   $_k3  = reverse($_k1);
   $_k3 =~ tr /[a-m][n-z]/[n-z][a-m]/;
   $_k3 =~ tr /[A-M][N-Z]/[N-Z][A-M]/;
   $_r1 = _xor( $_s,  "$_k1" );
   $_r2 = _xor( $_r1, "$_k2" );
   $_r3 = _xor( $_r2, "$_k3" );

   for ( $_d = 0 ; $_d < length($_r3) ; $_d++ ) {
      $_t[$_d] = sprintf( "%02x", ord( substr( $_r3, $_d, 1 ) ) );
   }

   $_e = join '', @_t;

   #    $_e =~ s/a/\./g;
   #    $_e =~ s/b/-/g;
   #    $_e =~ s/c/\+/g;
   #    $_e =~ s/d/\!/g;
   #    $_e =~ s/e/\=/g;
   #    $_e =~ s/f/\^/g;

   return reverse($_e);
}
```

</CODE>

[0037] For the present invention to properly work, however, the same effective encryption algorithm must be used on both server and client software.

[0038] In summary, known Anti-Spam mechanisms are commonly based upon filtering either patterns of characters, content type or heuristic learning of unwanted content. The present invention, instead, provides an open email box with an effective and simple locking mechanism. Hence, email that is wanted is delivered as the sender will have sought or obtained, by some known means including verbally or via written media, the required authorization code. The sender then derives the extended open (Calico'ed) email address, which will ensure the sender's email will properly accepted by the receiver, whereby unwanted emails will simply not arrive as they cannot be authenticated by the present authentication mechanism; e.g., the Calico Engine.

[0039] Alternative embodiments of the present invention may include using a plain text authentication code, numerical sequences, or some other cryptographic mechanism. Moreover, in yet other embodiments, the extended format may be altered as required by the particular application, such that the resultant email address is still RFC822 compliant. Those skilled in the art will readily be able to configure yet other alternative embodiments such that the authorization sequence is also changed on a daily basis, which embodiments are especially suitable "Web Based" implementations of the present invention, such as the "Calico'ed" email service provided by www.CalicoMail.com.

[0040] The following Figures illustrate just a few of the multiplicity of real world usage scenarios of the present invention. FIG. 5 illustrates usage of an embodiment of the present invention suitable for emailing between to relative lay computer users. FIG. 6 illustrates an embodiment of the present invention suitable for e-commerce applications where the e-commerce company has previously implemented the present invention. FIG. 7 illustrates an embodiment of the present invention adaptable and suitable for e-commerce applications where the e-commerce company does not implement the present invention. FIG. 8 illustrates an embodiment of the present invention suitable for correspondence initiated through a web page. Those skilled in the art will readily recognize a multiplicity of alternative and equivalent modifications, configurations, and steps to achieve an RFC822 compliant Anti-Spam mechanism in accordance with the principles. The present invention substantially reduces, if not completely eliminates, bulk emailing lists and address harvesting that currently plagues Internet email users.

[0041] Among the many alternative embodiments, the present invention may be configured as Server Software operable on any existing or future Operating System, as a Web based email service (such as CalicoMail.com), as an XML web services for constructing extended email addresses, or as Email clients with "Open Email Address Generation" according to the present invention built in.

[0042] FIG. 9 illustrates a typical computer system that, when appropriately configured or designed, can serve as a computer system in which the invention may be embodied. The computer system 900 includes any number of processors 902 (also referred to as central processing units, or CPUs) that are coupled to storage devices including primary storage 906 (typically a random access memory, or RAM), primary storage 904 (typically a read only memory, or ROM). CPU 902 may be of various types including microcontrollers and microprocessors such as programmable devices (e.g., CPLDs and FPGAs) and unprogrammable devices such as gate array ASICs or general purpose microprocessors. As is well known in the art, primary storage 904 acts to transfer data and instructions uni-directionally to the CPU and primary storage 906 is used typically to transfer data and instructions in a bi-directional manner. Both of these primary storage devices may include any suitable computer-readable media such as those described above. A mass storage device 908 may also be coupled bi-directionally to CPU 902 and provides additional data storage capacity and may include any of the computer-readable media described above. Mass storage device 908 may be used to store programs, data and the like and is typically a secondary storage medium such as a hard disk. It will be appreciated that the information retained within the mass storage device 908, may, in appropriate cases, be incorporated in standard fashion as part of primary storage 906 as virtual memory. A specific mass storage device such as a CD-ROM 914 may also pass data uni-directionally to the CPU.

[0043] CPU 902 may also be coupled to an interface 910 that connects to one or more input/output devices such as such as video monitors, track balls, mice, keyboards, microphones, touch-sensitive displays, transducer card readers, magnetic or paper tape readers, tablets, styluses, voice or handwriting recognizers, or other well-known input devices such as, of course, other computers. Finally, CPU 902 optionally may be coupled to an external device such as a database or a computer or telecommunications or Internet

network using an external connection as shown generally at 912. With such a connection, it is contemplated that the CPU might receive information from the network, or might output information to the network in the course of performing the method steps described herein.

[0044] Having fully described at least one embodiment of the present invention, other equivalent or alternative methods of implementing RFC822 compliant Anti-Spam mechanisms according to the present invention will be apparent to those skilled in the art. The invention has been described above by way of illustration, and the specific embodiments disclosed are not intended to limit the invention to the particular forms disclosed. The invention is thus to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the following claims.

I claim:

An Anti-Spam system comprising:

means for pre-pending email addresses with an authorization code thereby forming an extended email address;

means for recognizing said extended email address in an email; and

means for allowing the email to be delivered if said authorization code is recognized by said recognizing means.