```
In[83]:= (*Load the RSA package*)BeginPackage["SKPackages`RSA`"];

         RSAPrime::usage =
           "RSAPrime[digits] returns a prime number suitable for the RSA algorithm
             with the specified number of digits.";
         GenerateE::usage = "GenerateE[p, q] generates a random encoding
             exponent e that is coprime to (p-1)(q-1).";
         GenerateD::usage = "GenerateD[e, p, q] returns the multiplicative
             inverse of e mod((p-1)(q-1)).";
         StringToList::usage = "StringToList[text, n] converts a string
             into a list of numbers each less than n.";
         ListToString::usage = "ListToString[l] converts a list of numbers
             (produced by StringToList) back into the original string.";
         RSAEncodeNumber::usage = "RSAEncodeNumber[num, e, n] encrypts the number
             num using the RSA algorithm with exponent e and modulus n.";
         RSADecodeNumber::usage = "RSADecodeNumber[num, d, n] decrypts the number
             num using the RSA algorithm with exponent d and modulus n.";
         RSAEncode::usage = "RSAEncode[text, e, n] encrypts text by converting it into a
             list of numbers and using the RSA algorithm with exponent e and modulus n.";
         RSADecode::usage = "RSADecode[l, d, n] decrypts the list of numbers
             l using the RSA algorithm with exponent d and modulus n,
             and converts the result back into the original text.";

         Begin["`Private`"];

         GoodRSAPrimeQ[n_Integer, l_Integer] :=
           PrimeQ[n] && Module[{d = 1, m = (n - 1) / 2}, While[d ≤ l && ! PrimeQ[m / d], d++];
             d ≤ l];
         RSAPrime[digits_Integer] :=
           Module[{l, cand}, cand = Random[Integer, {10^(digits - 1), 10^digits - 1}];
            l = 10^Floor[Log[10., cand] / 30];
            If[EvenQ[cand], cand++];
            While[! GoodRSAPrimeQ[cand, l], cand += 2]; cand];
         GenerateE[p_Integer, q_Integer] :=
           Module[{res, n = p q, phi = (p - 1) (q - 1)}, res = Random[Integer, {p, n}];
            While[GCD[res, phi] ≠ 1, res = Random[Integer, {p, n}]]; res];
         GenerateD[e_Integer, p_Integer, q_Integer] := PowerMod[e, -1, (p - 1) (q - 1)];
         RSAEncodeNumber[num_Integer, e_Integer, n_Integer] := PowerMod[num, e, n] /; num < n;
         RSADecodeNumber[num_Integer, d_Integer, n_Integer] := PowerMod[num, d, n];
         ConvertString[str_String] := Fold[Plus[256 #1, #2] &, 0, ToCharacterCode[str]];
         StringToList[text_String, n_Integer] :=
           Module[{blockLength = Floor[N[Log[256, n]]], strLength = StringLength[text]},
             ConvertString /@ Table[StringTake[text, {i, Min[strLength, i + blockLength - 1]}],
               {i, 1, strLength, blockLength}]] /; n > 256;
         MakeList[0] = {};
         MakeList[num_Integer] := Append[MakeList[Quotient[num, 256]], Mod[num, 256]];
         ConvertNumber[num_Integer] := FromCharacterCode /@ MakeList[num];
         ListToString[l_List] := StringJoin[ConvertNumber /@ l];
         RSAEncode[text_String, e_Integer, n_Integer] :=
           RSAEncodeNumber[#, e, n] & /@ StringToList[text, n] /; n > 256;
         RSADecode[l_List, d_Integer, n_Integer] :=
           ListToString[RSADecodeNumber[#, d, n] & /@ l];

         End[];
```

```
EndPackage[];

(*Test the encryption and decryption throughput*)

(*Generate RSA keys*)
digits = 5;
p = SKPackages`RSA`RSAPrime[digits];
q = SKPackages`RSA`RSAPrime[digits];
n = p q;
e = SKPackages`RSA`GenerateE[p, q];
d = SKPackages`RSA`GenerateD[e, p, q];

(*Define the text to be encrypted and decrypted*)
text =
  "Sempre caro mi fu quest'ermo colle che da tanta now i am feeling very very freaked
    out because i do not know what is happening to my plenty know of code i
    got from the professor and i am about to fell very useless German poison,
    ego kill you one second! one second payne one second payne kk now we
    go speak ga then look.. or we go speak native Italian.. or sicilian";

(*Encrypt the text*)
encryptTime = AbsoluteTiming[encrypted = SKPackages`RSA`RSAEncode[text, e, n];][[1]];

(*Decrypt the text*)
decryptTime =
  AbsoluteTiming[decrypted = SKPackages`RSA`RSADecode[encrypted, d, n];][[1]];

(*Verify the decryption*)
verification = text === decrypted;

(*Calculate the size of the text in bytes*)textSize = StringLength[text];

(*Calculate encryption and decryption throughput*)
encryptionThroughput = textSize / encryptTime;
(*bytes per second*)decryptionThroughput = textSize / decryptTime;
(*bytes per second*)(*Print the results*)
Print["Encryption Time: ", encryptTime, " seconds"];
Print["Decryption Time: ", decryptTime, " seconds"];
Print["Encryption Throughput: ", encryptionThroughput, " bytes per second"];
Print["Decryption Throughput: ", decryptionThroughput, " bytes per second"];
Print["Verification: ", verification];
```

```
Encryption Time: 0.0044828 seconds

Decryption Time: 0.0023528 seconds

Encryption Throughput: 81422.3 bytes per second

Decryption Throughput: 155134. bytes per second

Verification: True
```