Integer Factorization

In [37]:
```
%pip install sympy
```

```
Requirement already satisfied: sympy in /opt/.qbraid/environments/qbraid_000000/pyen
v/lib/python3.11/site-packages (1.13.3)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /opt/.qbraid/environments/qbrai
d_000000/pyenv/lib/python3.11/site-packages (from sympy) (1.3.0)
Note: you may need to restart the kernel to use updated packages.
```

In [38]:
```
from sympy.ntheory import factorint
```

In [39]:
```
N=12
```

In [40]:
```
display(factorint(N))
```

```
{2: 2, 3: 1}
```

Factoring small numbers like 12 is easy, but when the number N to be factored gets larger, the problem becomes more difficult. For example, running factorint on a significantly larger number will cause a noticeable delay.

In [41]:
```
N = 3402823669209384634633740743176823109843098343
display(factorint(N))
```

```
{3: 2, 74519450661011221: 1, 5073729280707932631243580787: 1}
```

For even larger values of N, things become impossibly difficult, at least as far as we know. For example, the RSA Factoring Challenge, which was run by RSA Laboratories from 1991 to 2007, offered a cash prize of $100,000 to factor the following number, which has 309 decimal digits (and 1024 bits when written in binary). The prize for this number was never collected and its prime factors remain unknown.

In [42]:
```
RSA1024 = 135066410865995223349603216278805969938881475605667027524485143851526510
display(RSA1024)
```

```
1350664108659952233496032162788059699388814756056670275244851438515265106048595338339
4028715057190944179820728216447155137368041970396419174304649658927425623934102086
4383202110372958725762358509643110564073501508187510676594629205563685529475213500852
879416377328533906109750544334999811150056977236890927563
```

You do not get the integer factorization !

We need not bother running factorint on RSA1024, it wouldn't finish within our lifetimes.

The fastest known algorithm for factoring large integers is known as the number field sieve. As an example of this algorithm's use, the RSA challenge number RSA250, which has 250 decimal digits (and 829 bits in its binary representation), was factored using the number field sieve in 2020. The computation required thousands of CPU core-years, distributed across tens of thousands of machines around the world.

In [43]:
```
RSA250 = 21403246502407449612644230728393335630086147151447550177977549208814180234

p = 6413528947707158027879019017057738908482501474294344720811685963202453234463023
q = 3337202759497815655622601060535511422794076034476755466678452098702384172921003

display(RSA250 == p * q)
```

```
True
```

The security of the RSA public-key cryptosystem is based on the computational difficulty of integer factoring; an efficient algorithm for integer factoring would break it.

Greatest Common Divisor

The Greatest Common Divisor of two numbers is the largest integer that evenly divides both of them. This problem is easy for computers — it has roughly the same computational cost as multiplying the two input numbers together. The following code cell runs the `gcd` function from the Python `math` module on two numbers that are both quite a bit larger than RSA1024 in the blink of an eye. (In fact, RSA1024 is the GCD of the two numbers in this example.)

In [44]:
```
import math
K = 6413528947707158027879019017057738908482501474294344720811685963202453234463023
N = 4636759690183918349682239573236686632636353319755818421393667064929987310592347
M = 5056714874804877864225164843977749374751021379176083540426461360945653967249306

display(math.gcd(N**200 + 1, M**100 + M**2))
```

```
113
```

This is possible because we have very efficient algorithms for computing GCDs, the most well-known of which is Euclid's algorithm, discovered over 2,000 years ago.

Could there be a fast algorithm for integer factorization that we just haven't discovered yet, allowing large numbers like RSA1024 to be factored in the blink of an eye? The answer is yes. Although we might expect that an efficient algorithm for factoring as simple and elegant as Euclid's algorithm for computing GCDs would have been discovered by now, there is nothing that rules out the existence of a very fast classical algorithm for integer factoring, beyond the fact that we've failed to find one thus far. One could be discovered tomorrow — but don't hold your breath. Generations of mathematicians and computer scientists have searched, and factoring numbers like RSA1024 remains beyond our reach.

Modular Exponent

Here's a code cell that computes a modular exponent for fairly large input numbers.

In [45]:
```
K = 6413528947707158027879019017057738908482501474294344720811685963202453234463023
N = 4636759690183918349682239573236686632636353319755818421393667064929987310592347
M = 5056714874804877864225164843977749374751021379176083540426461360945653967249306

display(pow(N, K**10 + 1, M**10 + 1))
```

```
542080890895977406028898753671977045621731891209489822571389293690904902920587526839
101677349627301627291932938266953312714140538161831995587114781113307168311397674110
656056434861136283920974891097411279989438546490061766446832927171765506749534185882
270482964957955172448315526420831404874246938784702040798753787302737903651594235520
642557610494239445239004008106904185852521798504471201802899139463545887022535847609
923204071461037767259867972001546270283393458045915162624313117846527188469526029419
850301047820703955137065755680198054632893968338978289310193797702316002702082461211
835098117229999778486236491442678439570530462594698162087656590363032142671356460163
661500352645055945098825705426557667378399638968422594650395076776491176591250047897
112040771204383272670653999920293573293682040053497085533829465336908163404880411532
025618221114554409481610762912411274224922631935353684314325067555979174335589299305
107848828529292717802343840258305483983210950378147159650007295910770702006578255586
132525389117446847982845507070180002421259314497140083214078606362066522433902774604
897849658926560243028647522924050073039503024416737376077322572409015600404935259463
586702793546596241476445637358529095699930579496461756154827056521608042123837065251
899742149917872205861725656405329075815384157474334102023072366510556838392333552498
126239531483729229135121357985323380066454395969757066896893254510539120089434512240
390112685580833929289214461803464764059412963889180934743351695405425573537411422968
746725366150940432908103846753697648531400545779464279324072890771674811358262304789
692729952536480346178433989934901133184236611983498525476702279781754198284213603879
734604552454344683760969196144634689803448196016236990654720785174070332559527372431
350986679813979186273894141243508849079047399254265525618735905264899522657981378010
176370063278417731899859396012397737606518864284971440207182133265609924659519227354
812457930895436937916679137550436617796336716735773420425013452390101205372841450566
208371948327714999829734560085020796859461161050205811975405245805961615346032178414
153999005638080857111717267738993899715942577694906503713313819724716111512765958510
947007678421284668984344987429452828882911743581872753258319074174268660109058053405
028959201676113567960336490407230556944838274187036624439739658256545328264140219540
702398153069175149689822263394219823027090709891325399565197861154506619395018871824
565319783779159642632956094134775393965042961144724557707884693172237261636755000635
562036027645556808051209903517917541675614323395093421473662195003990250664422144399
843534146045247646617141997255839564468997146574790767489679082209766030710906293195
769468403397841295256759959348868752901152635910616980955597966740186672327793120151
017763030213384793215827899031268233454937765252062693583573954771298331432079857389
422907643719070485636439632779515148427895652664859726008482869730842716970425154667
383258360970545657528153372217802910319225676534350464511282690207396022340798962448
167266870397240841466589440334718167916077421753615336442243684072155720257234802193
251187770863989986604206492515099787011561013009192330871637249150560023463832856025
299856125543587286707919489881983889831150243019029551221559467757627524822095467570
419971288019688765012640181057427011829561921669300124768924526437256442335583042108
727920426483427629079008969263698419817693076459778852962041082118852278201677791079
6700483449826
```

It's a little bit slower than computing GCDs, but it's still fast.

A different example, outside of the domain of computational number theory, arose in the previous lesson. For the post-processing step of Simon's algorithm, we need to compute the null space modulo 2 of an n×m matrix of binary values (so here the input length is nm bits in total). We can do this using Gaussian elimination with O(nm min{n,m}) elementary operations, which is O(n**3) elementary operations if m=O(n). Even for a 1000×1000 binary matrix, which is a million bits of input, the computation time is on the order of seconds.

In [46]:
```
%pip install galois
```

```
Requirement already satisfied: galois in /opt/.qbraid/environments/qbraid_000000/pye
nv/lib/python3.11/site-packages (0.4.3)
Requirement already satisfied: numpy>=1.21.0 in /opt/conda/lib/python3.11/site-packa
ges (from galois) (1.26.4)
Requirement already satisfied: numba<0.61,>=0.55 in /opt/.qbraid/environments/qbraid
_000000/pyenv/lib/python3.11/site-packages (from galois) (0.60.0)
Requirement already satisfied: typing-extensions>=4.0.0 in /opt/conda/lib/python3.1
1/site-packages (from galois) (4.12.2)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /opt/.qbraid/environmen
ts/qbraid_000000/pyenv/lib/python3.11/site-packages (from numba<0.61,>=0.55->galois)
(0.43.0)
Note: you may need to restart the kernel to use updated packages.
```

The galois library is a Python 3 package that extends NumPy arrays to operate over finite fields.

In [47]:
```
import galois
```

In [48]:
```
GF = galois.GF(2)
```

In [49]:
```
N, n = 1000, 1000
```

In [50]:
```
A = GF.Random((N, n))
```

In [51]:
```
B = A.null_space()
```

In [52]:
```
display(B)
```

```
GF([[0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
     0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
     0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1,
     0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1,
     1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1,
     0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0,
     1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0,
     0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
     0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
     0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
     0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
     1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0,
     1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1,
     0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1,
     1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0,
     1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
     1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
     0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
     0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
     1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
     1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1,
     1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
     0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
     1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1,
     1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1,
     1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
     0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0,
     1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1,
     1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0,
     1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0,
     0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
     0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,
     1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
     0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1,
     1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
     1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1,
     1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
     1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0,
     0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
     1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1]], order=2)
```

Construction of a Toffoli Gate

In [53]:
```
%pip install qiskit[visualization]
```

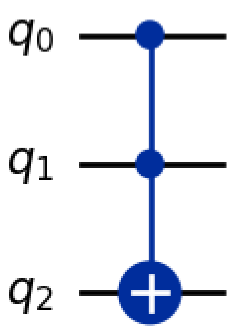
```
Requirement already satisfied: qiskit[visualization] in /opt/.qbraid/environments/qb
raid_000000/pyenv/lib/python3.11/site-packages (1.3.0)
Requirement already satisfied: rustworkx>=0.15.0 in /opt/.qbraid/environments/qbraid
_000000/pyenv/lib/python3.11/site-packages (from qiskit[visualization]) (0.15.1)
Requirement already satisfied: numpy<3,>=1.17 in /opt/conda/lib/python3.11/site-pack
ages (from qiskit[visualization]) (1.26.4)
Requirement already satisfied: scipy>=1.5 in /opt/.qbraid/environments/qbraid_00000
0/pyenv/lib/python3.11/site-packages (from qiskit[visualization]) (1.14.1)
Requirement already satisfied: sympy>=1.3 in /opt/.qbraid/environments/qbraid_00000
0/pyenv/lib/python3.11/site-packages (from qiskit[visualization]) (1.13.3)
Requirement already satisfied: dill>=0.3 in /opt/.qbraid/environments/qbraid_000000/
pyenv/lib/python3.11/site-packages (from qiskit[visualization]) (0.3.9)
Requirement already satisfied: python-dateutil>=2.8.0 in /opt/conda/lib/python3.11/s
ite-packages (from qiskit[visualization]) (2.9.0)
Requirement already satisfied: stevedore>=3.0.0 in /opt/.qbraid/environments/qbraid_
000000/pyenv/lib/python3.11/site-packages (from qiskit[visualization]) (5.4.0)
Requirement already satisfied: typing-extensions in /opt/conda/lib/python3.11/site-p
ackages (from qiskit[visualization]) (4.12.2)
Requirement already satisfied: symengine<0.14,>=0.11 in /opt/.qbraid/environments/qb
raid_000000/pyenv/lib/python3.11/site-packages (from qiskit[visualization]) (0.13.0)
Requirement already satisfied: matplotlib>=3.3 in /opt/conda/lib/python3.11/site-pac
kages (from qiskit[visualization]) (3.9.3)
Requirement already satisfied: pydot in /opt/.qbraid/environments/qbraid_000000/pyen
v/lib/python3.11/site-packages (from qiskit[visualization]) (3.0.3)
Requirement already satisfied: Pillow>=4.2.1 in /opt/conda/lib/python3.11/site-packa
ges (from qiskit[visualization]) (11.0.0)
Requirement already satisfied: pylatexenc>=1.4 in /opt/.qbraid/environments/qbraid_0
00000/pyenv/lib/python3.11/site-packages (from qiskit[visualization]) (2.10)
Requirement already satisfied: seaborn>=0.9.0 in /opt/.qbraid/environments/qbraid_00
0000/pyenv/lib/python3.11/site-packages (from qiskit[visualization]) (0.13.2)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.11/site-pa
ckages (from matplotlib>=3.3->qiskit[visualization]) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.11/site-packag
es (from matplotlib>=3.3->qiskit[visualization]) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.11/site-p
ackages (from matplotlib>=3.3->qiskit[visualization]) (4.55.2)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/conda/lib/python3.11/site-p
ackages (from matplotlib>=3.3->qiskit[visualization]) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.11/site-pac
kages (from matplotlib>=3.3->qiskit[visualization]) (24.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/lib/python3.11/site-pa
ckages (from matplotlib>=3.3->qiskit[visualization]) (3.2.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.11/site-packages
(from python-dateutil>=2.8.0->qiskit[visualization]) (1.16.0)
Requirement already satisfied: pandas>=1.2 in /opt/conda/lib/python3.11/site-package
s (from seaborn>=0.9.0->qiskit[visualization]) (2.2.3)
Requirement already satisfied: pbr>=2.0.0 in /opt/.qbraid/environments/qbraid_00000
0/pyenv/lib/python3.11/site-packages (from stevedore>=3.0.0->qiskit[visualization])
(6.1.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /opt/.qbraid/environments/qbrai
d_000000/pyenv/lib/python3.11/site-packages (from sympy>=1.3->qiskit[visualization])
(1.3.0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.11/site-packag
es (from pandas>=1.2->seaborn>=0.9.0->qiskit[visualization]) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.11/site-pack
```

```
ages (from pandas>=1.2->seaborn>=0.9.0->qiskit[visualization]) (2024.2)
Note: you may need to restart the kernel to use updated packages.
```

In [54]:
```
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector, Operator
from qiskit.visualization import array_to_latex
```

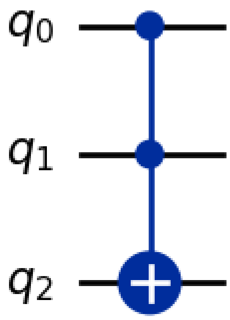
In [55]:
```
from sympy import latex
```

In [56]:
```
Toffoli = QuantumCircuit(3)
# Replace the ? to create the Circuit that is depicted below.
Toffoli.ccx(0, 1, 2)

display(Toffoli.draw('mpl'))
U = Operator(Toffoli)

display(array_to_latex(U))
```




```
$$ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \end{bmatrix} $$
```

In [57]:
```
# Replace the ?
BuiltinToffoli = QuantumCircuit(3)
BuiltinToffoli.ccx(0, 1, 2)
display(BuiltinToffoli.draw('mpl'))
BITU = Operator(BuiltinToffoli)

display(array_to_latex(BITU))
```



$$ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \end{bmatrix} $$

End of Notebook