Supporting information to

Storing the portrait of *Antoine de Lavoisier* in a Single Macromolecule

Eline Laurent,[1] Jean-Arthur Amalian,[2] Thibault Schutz,[1] Kevin Launay,[2] Jean-Louis Clément,[2] Didier Gigmes,[2] Alexandre Burel,[3] Christine Carapito,[3] Laurence Charles,[2] Marc-André Delsuc,[4]* and Jean-François Lutz[1]**

[1] Université de Strasbourg, CNRS, Institut Charles Sadron UPR22, 23 rue du Loess, 67034 Strasbourg Cedex 2, France. E-mail : jflutz@unistra.fr

[2] Aix Marseille Université, CNRS, Institute for Radical Chemistry, UMR 7273, 23 Av Escadrille Nomandie-Niemen, 13397 Marseille Cedex 20, France.
E-mail : laurence.charles@univ-amu.fr

[3] Université de Strasbourg, CNRS, Laboratoire de Spectrométrie de Masse BioOrganique (LSMBO), IPHC, 23 rue du Loess, 67034 Strasbourg Cedex 2, France

[4] Institut de Génétique et de Biologie Moléculaire et Cellulaire, INSERM, U596, CNRS, UMR7104, Université de Strasbourg, 1 rue Laurent Fries, 67404, Illkirch-Graffenstaden, France.
E-mail : madelsuc@unistra.fr

A. Experimental procedures

A.1. Reagents for automated phosphoramidite synthesis. Anhydrous acetonitrile (phosphoramidite diluent & dry washings, ChemGenes), acetonitrile (≥99.9 %, washings, Roth), activation reagent (0.25 M 5 ethylthio tetrazole in MeCN, ChemGenes), Cap A (acetic anhydride/pyridine/THF, ChemGenes), Cap B (10 % N-methylimidazole in THF, ChemGenes), DMT removal reagent (3 w% TCA in DCM, Roth), drying traps (small, 10 - 15 mL, ChemGenes), polystyrene dT support (1 µmol for ABI3900, Distribio), glen-pak DNA purification cartridge (10 nmole - 1.0 µmole, Glen Reasearch) and oxidizing solution (0.02 M iodine/pyridine/H_2O/THF, ChemGenes) were used as received. The nucleoside phosphoramidites (Figure S1) Ac-dC-CE phosphoramidite, dA-CE phosphoramidite dG-CE phosphoramidite, 5-Br-dU-CE phosphoramidite, 5-I-dU-CE phosphoramidite, 2′-F-G-CE phosphoramidite, 8-Br-dG-CE phosphoramidite, 2'-DeoxyNebularine-CE phosphoramidite (Purine) and 2'-F-Ac-C-CE phosphoramidite (Glen Research) were also used as received. The phosphoramidites monomers **M_1-M_8** and the linker RISC2 were prepared following previously-reported procedures.[1-2]All the phosphoramidites monomers were stored in the freezer at −18 °C.

A.2. Automated solid-phase synthesis. The polymer was synthesized under argon in rigorously dry conditions by automated solid-phase phosphoramidite method on an ABI DNA Synthesizer (Applied BioSystems 3900). The typical method involves four steps *(i)* deprotection, *(ii)* coupling, *(iii)* oxidation and *(iv)* capping, as described previously.[3] The phosphoramidite monomers **M_1-M_8**, the linker RISC2 and the nucleoside phosphoramidites A, C, G, B, I, F, R, P and D were dissolved in anhydrous acetonitrile under argon. A concentration of 50 mM was used for all monomers except for **M_4-M_8** (60 mM). The monomers were placed in the synthesizer with all the needed reagents (activation reagent, Cap A, Cap B, DMT removal reagent) and primed twice. The polystyrene solid support dT (1 µmol scale) was placed in the synthesizer and the sequences were performed with DMT-On mode. Once the synthesis was done the solid support column was removed from the synthesizer and the DMT-protected sequence was cleaved from the support. A solution of 28% ammonia and methylamine (1/1, v/v) was used to cleave the DMT-protected polymers from the solid-support at RT for 30 min and purified on a glen-pak DNA purification cartridge. This procedure permits to separate the DMT-terminated targeted structures from the truncated sequences deactivated by the capping reaction. Then, the terminal DMT moiety of the desired sequence-coded polymers was removed directly on the glen-pak column and washed out by solvent elution. The isolated polymer solution was lyophilized and obtained as a white powder in 44% yield.

A.3. Code for display and compression. Code deposited at github.com/delsuc/Antoine_de_Lavoisier

```
import numpy as np
import matplotlib.pyplot as plt

# Some basic tools
# we use the following conventions:
#    Signal is the binary signal to code/decode          - coded into a np.array
#    Message is the molecular organisation of the signal  - coded into a list
# converters
Codon = 3  # number of bits carried by the monomers    3 ⇒ 8 monomers M1...M8

def decodon(lett, code=Codon):
    """
    given a letter returns bit pattern in given code
    decodon("M1", 3) == [0 0 0]
    decodon("M3", 2) == [1 0]
    """
    n = int(lett[1:])-1
    form = "{:0>%db}"%(code)
    s = form.format(n)
    return [int(l) for l in s]  # make it list
def codon(pat):
    """
    given a bit pattern, returns its name in letter
    [0 0 0] = "M1"
    [1 0] = "M3"   etc...
    """
    s = sum([(2**i) * pat[-1-i] for i in range(len(pat))] )
    return 'M%d'%(s+1)

def sig2msg(signal, code=Codon):
    "code a signal into a message"
    return [codon(signal[i:i+code]) for i in range(0,len(signal),code)]
def msg2sig(msg, code=Codon):
    "decode a message into a signal"
    return np.array(sum(map(decodon,msg),[]))

# checksun handling
def addcs(argsignal):
    """this function takes a signal, as a list of bits and add on cs at the end
    """
    signal = list(argsignal)
    cs = sum(signal)%2
    signal.append(cs)
    return signal
def checkcs(argsignal):
    signal = list(argsignal)
    cs = signal.pop()
    if cs != sum(signal)%2:
        print('**WARNING** the checsum is not valid')
    else:
        print('valid checksum')
    return np.array(signal)
# tools on pictures
def pcheck(pict):
    "verifies that pict (whatever format) is correct, and return a formated pict array"
    p = np.array(pict)
    if len(p.shape) == 1:
        p = p.reshape(SizeH,SizeW)
    else:
        p.shape != (SizeW, SizeH)
#        print(p.shape)
    return p
def draw( pict ):
    "draws a pixelated picture"
    plt.figure()
    plt.imshow(pcheck(pict), cmap="Greys")
```

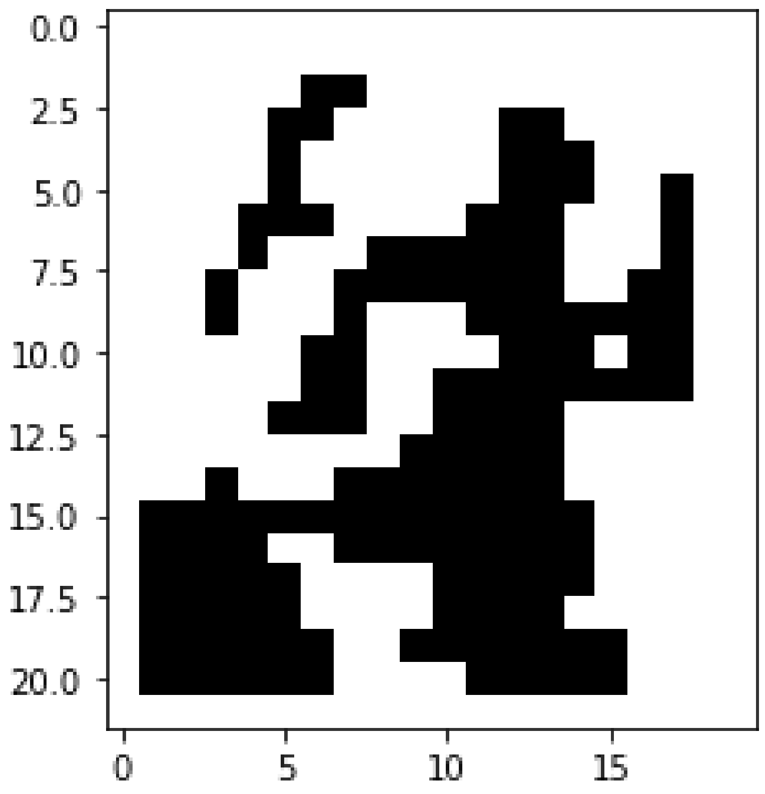
```
# exemple given
sig = [1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0]
print('signal:\t\t',sig)
sigcs = addcs(sig)
print('signal +cs :\t',sigcs)
msg = sig2msg(sigcs)
print('coded message:\t',msg)
```

```
signal:      [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
signal +cs :     [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
coded message:   ['M8', 'M8', 'M7', 'M1', 'M1', 'M1']
```

```
# the following code implements an arithmetic compresser
# based on the code from "Project Nayuki", https://www.nayuki.io/page/reference-arithmetic-coding
# dowloaded en march 11 2020
import io
import arithm_coding.arithmeticcoding  as ar
class BitInputStream(object):
    "Constructs a fake bit input stream"
    def __init__(self, inp,nbits):
        "inp is a list of 1 and 0"
        self.inp = inp
        self.nbits = nbits
    def read(self):
        try:
            return self.inp.pop(0)
        except:
            return -1
    def _repr_(self):
        self.inp._repr_()
class BitOutputStream(object):
    "Constructs a fake bit output stream"
    def __init__(self,nbits):
        self.out = []
        self.nbits = nbits
    def write(self, b):
        "Writes a bit to the stream. The given bit must be 0 or 1."
        if b not in (0, 1):
            raise ValueError("Argument must be 0 or 1")
        self.out.append(b)
    def close(self):
        pass
#        print(self.out)
    def _repr_(self):
        self.out._repr_()
# Code
class ARCoder():
    def __init__(self,nbits, verbose=True):
        if nbits <2 or nbits>8 :
            raise ValueError('N should be a value between 2 and 8')
        self.nbits = nbits
        self.sz = 2**nbits
        self.sig = None
        self.verbose = verbose
    def set_signal(self, signal):
        "to be called after initialization for coding"
        self.sig = self.align(signal)
        self.comp_frequencies()
    # compute frequencies
    def set_freq(self, freqs):
        "to be called after initialization for decoding - freqs is the frequency table computed by the coder"
        self.freqs = freqs
    def align(self, sig):
        "complete signal with trailing 0 so length is a multiple of 2^nbits"
        asig = [s for s in sig]
        while len(asig) % self.sz != 0:
            asig.append(0)
        return asig
    def comp_frequencies(self):
        "compute the frequency table from the signal"
        freqs = ar.SimpleFrequencyTable([0] * (self.sz+1))
        for i in range(0,len(self.sig), self.nbits):
            symb = 0
            for j in range(self.nbits):
                if i+j < len(self.sig):
                    v = self.sig[i+j]
                    symb = 2*symb + v
                else:
                    break
            freqs.increment(symb)
        freqs.increment(self.sz)  # marks end of stream
        self.freqs = freqs
    def compress(self):
        "return the compressed signal from the signal"
        out = open('toto','wb')
        bits = ar.BitOutputStream(out)
        coder = ar.ArithmeticEncoder(32,bits)
        for i in range(0,len(self.sig), self.nbits):
            symb = 0
            for j in range(self.nbits):
                if i+j < len(self.sig):
                    v = self.sig[i+j]
                    symb = 2*symb + v
                else:
                    break
            if self.verbose: print(symb,end=' ')
            coder.write(self.freqs, symb)
        coder.write(self.freqs, self.sz)     # marks end of stream
        coder.finish()
        bits.close()
        res = []
        bylst = open('toto','rb').read()
        for by in bylst:
            for _ in range(8):
                res.append(by%2)
                by = by//2
        return res
    def decomp(self, inp):
        inp = ar.BitInputStream(open('toto','rb'))
        dec = ar.ArithmeticDecoder(32,inp)
        res = []
        while True:
            symb = dec.read(self.freqs)
            if symb == self.sz:          # EOF symbol
                break
            if self.verbose: print(symb, end=' ')
            # decode symb
            bb = [0]*self.nbits
            for j in range(self.nbits):
                if symb%2 == 1:
                    bb[self.nbits-j-1] = 1
                symb = symb//2
            for b in bb:
                res.append(b)
#           res.append(symb)
        return res
def ARcode(image, N):
    "a minimum coder"
    C = ARCoder(N, verbose=False)
    C.set_signal(image)
    #print(C.freqs.frequencies)
    s = C.compress()
    return s
```

A.4. The Lavoisier picture.

```
Lavoisier = np.array
    ([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0],
      [0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0],
      [0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0],
      [0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0],
      [0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0],
      [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0],
      [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
      [0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
      [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
      [0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
      [0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
      [0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
      [0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
      [0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
SizeW, SizeH = (20, 22)  # size of the picture
draw(Lavoisier)
```



png

A.5. Compression and coding of the Lavoisier picture.

```
stream = Lavoisier.ravel()
lenstream = len(stream)
print('raw bit stream:\n', stream, '\nlength:',lenstream)
```

```
raw bit stream:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 1 1 1 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0
 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1 0 0 0 0 0 1 0
 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0
 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0
 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1
 1 1 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1
 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
length: 440
```

```
streamcs = np.array(addcs(stream))              # add the checksum
lenstreamcs = len(streamcs)
print('stream with added checksum:\n', streamcs, '\nlength:',lenstreamcs)
```

```
stream with added checksum:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 1 1 1 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0
 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1 0 0 0 0 0 1 0
 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0
 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0
 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1
 1 1 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1
 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
length: 441
```

```
# set-up the coder
N = 8                              # we'll code on 8bits
C = ARCoder(N, verbose=False)      # create the code
C.set_signal(streamcs)             # set the signal - computes the frequency table
freq_table = C.freqs               # and store it
# print("frequency table:\n",freq_table.frequencies)     # uncomment to print the frequency
table

# do the compression
compstream = np.array(C.compress())
complength = len(compstream)
print('compressed stream:\n', compstream, '\nlength:', complength, 'bits' )
```

```
print ("that's a %.0f%% reduction compared to non compressed"%(100*(1-
complength/len(streamcs))))
```

```
compressed stream:
 [0 0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 0 1 0 0 1 1 0 1 1 1
 1 1 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 1
 1 1 0 1 0 0 0 1 0 0 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 0 1 1 0 1 0 0 1 0 0 1
 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 0 0 1 1 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 1
 1 1 1 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 1 1 1 1 1 1 0 0 1 0 0 1 0 1 1 1 1 0 0
 0 0 0 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 0 0 1 0 1 1 1 0 1 0 0 0
 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 0 0 0 1 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1
 0 1 1 1 1]
length: 264 bits
that's a 40% reduction compared to non compressed
```

A.6. Transcription into the chemical alphabet.

```
msgLavoisier = sig2msg(compstream)
print('message:')
print(*msgLavoisier, sep='-')
print('length:', len(msgLavoisier), 'monomers')
```

message:

$M_1{\cdot}M_1{\cdot}M_2{\cdot}M_7{\cdot}M_7{\cdot}M_4{\cdot}M_2{\cdot}M_8{\cdot}M_8{\cdot}M_3{\cdot}M_4{\cdot}M_4{\cdot}M_8{\cdot}M_7{\cdot}M_4{\cdot}M_1{\cdot}M_7{\cdot}M_4{\cdot}M_2{\cdot}M_7{\cdot}M_7{\cdot}M_5{\cdot}M_6{\cdot}M_5{\cdot}M_8$-$M_6{\cdot}M_1{\cdot}M_5{\cdot}M_6{\cdot}M_8{\cdot}M_6{\cdot}M_5{\cdot}M_8{\cdot}M_6{\cdot}M_6{\cdot}M_2{\cdot}M_2{\cdot}M_3{\cdot}M_7{\cdot}M_5{\cdot}M_1{\cdot}M_6{\cdot}M_5{\cdot}M_4{\cdot}M_2{\cdot}M_7{\cdot}M_5{\cdot}M_2{\cdot}M_4{\cdot}M_8$-$M_6{\cdot}M_1{\cdot}M_8{\cdot}M_3{\cdot}M_1{\cdot}M_8{\cdot}M_8{\cdot}M_5{\cdot}M_5{\cdot}M_6{\cdot}M_8{\cdot}M_1{\cdot}M_2{\cdot}M_8{\cdot}M_1{\cdot}M_8{\cdot}M_7{\cdot}M_2{\cdot}M_8{\cdot}M_3{\cdot}M_2{\cdot}M_4{\cdot}M_6{\cdot}M_1{\cdot}M_8$-$M_8{\cdot}M_5{\cdot}M_8{\cdot}M_8{\cdot}M_4{\cdot}M_1{\cdot}M_8{\cdot}M_4{\cdot}M_4{\cdot}M_4{\cdot}M_2{\cdot}M_6{\cdot}M_8$

length: 88 coded monomers (without RISC2 and mass tags)

$M_1{\cdot}M_1{\cdot}M_2{\cdot}M_7{\cdot}M_7{\cdot}M_4{\cdot}M_2{\cdot}M_8$-*RISC2*-$M_8{\cdot}M_3{\cdot}M_4{\cdot}M_4{\cdot}M_8{\cdot}M_7{\cdot}M_4{\cdot}M_1{\cdot}$D-*RISC2*-$M_7{\cdot}M_4{\cdot}M_2{\cdot}M_7{\cdot}M_7{\cdot}M_5{\cdot}M_6{\cdot}M_5{\cdot}$P-*RISC2*-$M_8{\cdot}M_6{\cdot}M_1{\cdot}M_5{\cdot}M_6{\cdot}M_8{\cdot}M_6{\cdot}M_5{\cdot}$R-*RISC2*-$M_8{\cdot}M_6{\cdot}M_6{\cdot}M_2{\cdot}M_2{\cdot}M_3{\cdot}M_7{\cdot}M_5{\cdot}$F-*RISC2*-$M_1{\cdot}M_6{\cdot}M_5{\cdot}M_4{\cdot}M_2{\cdot}M_7{\cdot}M_5{\cdot}M_2{\cdot}$I-*RISC2*-$M_4{\cdot}M_8{\cdot}M_6{\cdot}M_1{\cdot}M_8{\cdot}M_3{\cdot}M_1{\cdot}M_8{\cdot}$B-*RISC2*-$M_8{\cdot}M_5{\cdot}M_5{\cdot}M_6{\cdot}M_8{\cdot}M_1{\cdot}M_2{\cdot}M_8{\cdot}$G-*RISC2*-$M_1{\cdot}M_8{\cdot}M_7{\cdot}M_2{\cdot}M_8{\cdot}M_3{\cdot}M_2{\cdot}M_4{\cdot}$A-*RISC2*-$M_6{\cdot}M_1{\cdot}M_8{\cdot}M_8{\cdot}M_5{\cdot}M_8{\cdot}M_8{\cdot}M_4{\cdot}$C-*RISC2*-$M_1{\cdot}M_8{\cdot}M_4{\cdot}M_4{\cdot}M_4{\cdot}M_2{\cdot}M_6{\cdot}M_8$-T.

Total length: 108 residues (88 coded monomers + 10 RISC2 + 10 mass tags)

B. Measurements and analyses

B.1. Size exclusion chromatography (SEC). Size exclusion chromatography was carried out on a DIONEX HPLC system, Ultimate 3000 (degasser, pump, auto sampler) equipped with 4 Shodex OH-pak columns 30 cm (802.5HQ, 804HQ, 806HQ, 807HQ) and a guard column (from 500 to 100 000 000 g·mol^{-1}). The eluent used as a solvent was 60% water (Millipore quality) + 40% acetonitrile (HPLC grade) + 0.1 M $NaNO_3$ with a flow rate of 0.5·mL min^{-1}. The polymer was diluted in the same solvent during 12 h and filtered on Dyanagard filter (cellulose ester) 0.2 µm. It was detected with the help of a differential refractometer OPTILAB rEX (Wyatt Techn.) and a light scattering detector DAWN HELEOS II (Wyatt Techn.). The reported M_n, M_p and *Đ* values were obtained from light scattering using a dn/dc value of 0.107 mL·g^{-1}.

B.2. Mass spectrometry. For sample preparation, reagents were from Sigma Aldrich (St Louis, MO) and solvents (HPLC grade) were from Carlo Erba (Peypin, France). The polymer was dissolved in H_2O/ACN (50:50, v/v) solution containing 0.1% formic acid, further diluted (1/10) in methanol and introduced at a 10 µL min^{-1} flow rate in the electrospray ionization (ESI) source of a hybrid QTOF mass spectrometer Synapt G2-HDMS (Waters, Manchester, UK). The ESI source was operated in the negative mode (capillary voltage: –2.27 kV) under a desolvation gas (N_2) flow of 100 L h^{-1} heated at 35°C. The cone voltage was set to –60 V. Collision-induced dissociation (CID) was performed in the ion trap device using argon as the collision gas after selection of precursor ions in the quadrupole mass analyzer of the instrument. Instrument control, data acquisition and data processing of all experiments were achieved with the MassLynx 4.1 programs provided by Waters.

MALDI-TOF MS experiments were carried out using a Bruker Autoflex (Bruker Daltonics, Leipzig, Germany) equipped with a nitrogen laser emitting at 337 nm, a single-stage pulsed ion extraction source and dual microchannel plate detectors. Data acquisition was performed in the linear mode and the MALDI source was operated in the negative ion mode, using a laser frequency of 10 Hz and an accelerating voltage of 20 kV. The delay time used in delayed extraction mode was optimized to 100 ns. Mass spectra were acquired using a 55% laser fluence by performing a series of 400 shots while moving the laser around the sample surface. Ions below *m/z* 500 were deflected to suppress high intensity matrix signals. MALDI sample preparation was inspired from a protocol for mass analysis of oligonucleotides,[4] using 3-hydroxypicolinic acid (3-HPA) as the matrix. A saturated solution of 3-HPA was prepared in a water/acetonitrile (75/25, v/v) binary mixture. A two-step MALDI sample preparation was used, with 2 µL of the polymer solution first deposited and left dried on the sample plate before 1 µL of the 3-HPA matrix solution was added on top and dried under atmospheric conditions.

B.3. Automated sequencing by MS-DECODER. An algorithm for RISC2-based poly(phosphodiester)s was implemented in MS-DECODER as previously described.[2] This algorithm works for sequences of 8 entities per fragmentation spectrum, and can therefore adjust to entities of one, two or three bits as it is the case in the present work. This adjustment is possible thanks to a table of all theoretical *m/z* values for each composition and tag. Based on this table, it is possible to compute the theoretical masses to search for every possible combination of N-bits entities and charge state. Although the number of combinations increases with the value of N, the algorithm manages to automatically detect a consequent number of impossible combinations, prevents their full computation and in memory storage and thus tends to reduce the algorithmic complexity of MS-DECODER. After generating the possible combinations, the next step is to check which combination(s) fit the experimental MS/MS spectra the best. This is done in a single parsing of the spectrum, by storing each fragment matching a theoretical fragment from one of the combinations. At the end, combinations are ordered by the number of matching *m/z* values. Only the 10 best combinations are kept and returned to the user. Through the graphical interface, the user can visualize the output of possible combinations for each MS/MS spectrum, thus sub-segment, and can select the most appropriate match.

B.4. Decoding the chemical message.

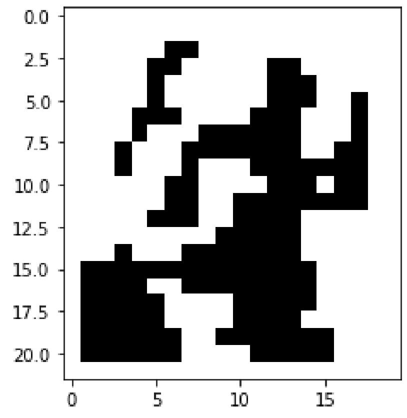
```
# first decode the message into a compressed signal
decstream = msg2sig(msgLavoisier)
print('decoded signal:\n',decstream)
```

```
decoded signal:
 [0 0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 0 1 0 0 1 1 0 1 1 1
 1 1 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 1
 1 1 0 1 0 0 0 1 0 0 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 0 1 1 0 1 0 0 1 0 0 1
 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 0 0 1 1 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 1
 1 1 1 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 1 1 1 1 1 1 0 0 1 0 0 1 0 1 1 1 1 0 0
 0 0 0 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 0 0 1 0 1 1 1 0 1 0 0 0
 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 0 0 0 1 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1
 0 1 1 1 1]
```

```
# decode the signal
D = ARCoder(N, verbose=False)          # create the decoder
D.set_freq(freq_table)                 # load the frequency table
decoded = D.decomp(decstream)
decoded = np.array(decoded[:lenstreamcs])
decoded = checkcs(decoded)
print('decoded signal\n',decoded)
```

```
valid checksum
decoded signal
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 1 1 1 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0
 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1 0 0 0 0 0 1 0
 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 0 1 1 0 0 0 0
 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0
 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1
 1 1 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1
 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
draw(decoded)
```



C. Additional data and figures

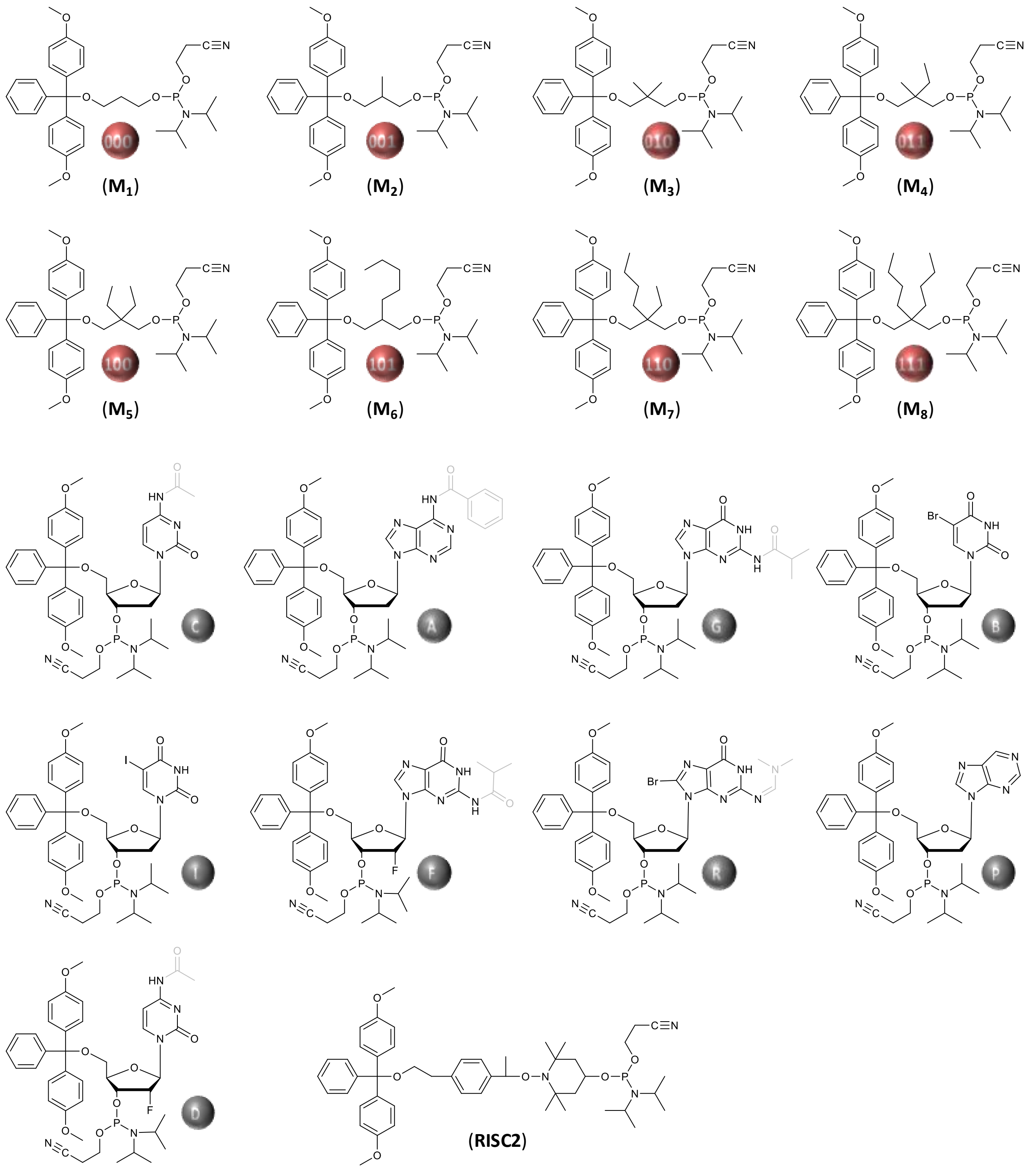


Figure S1. Molecular structure of the different phosphoramidite monomers used in this work. The protecting groups displayed in light grey on the purine and pyrimidine rings are removed during cleavage, leading to primary amines.

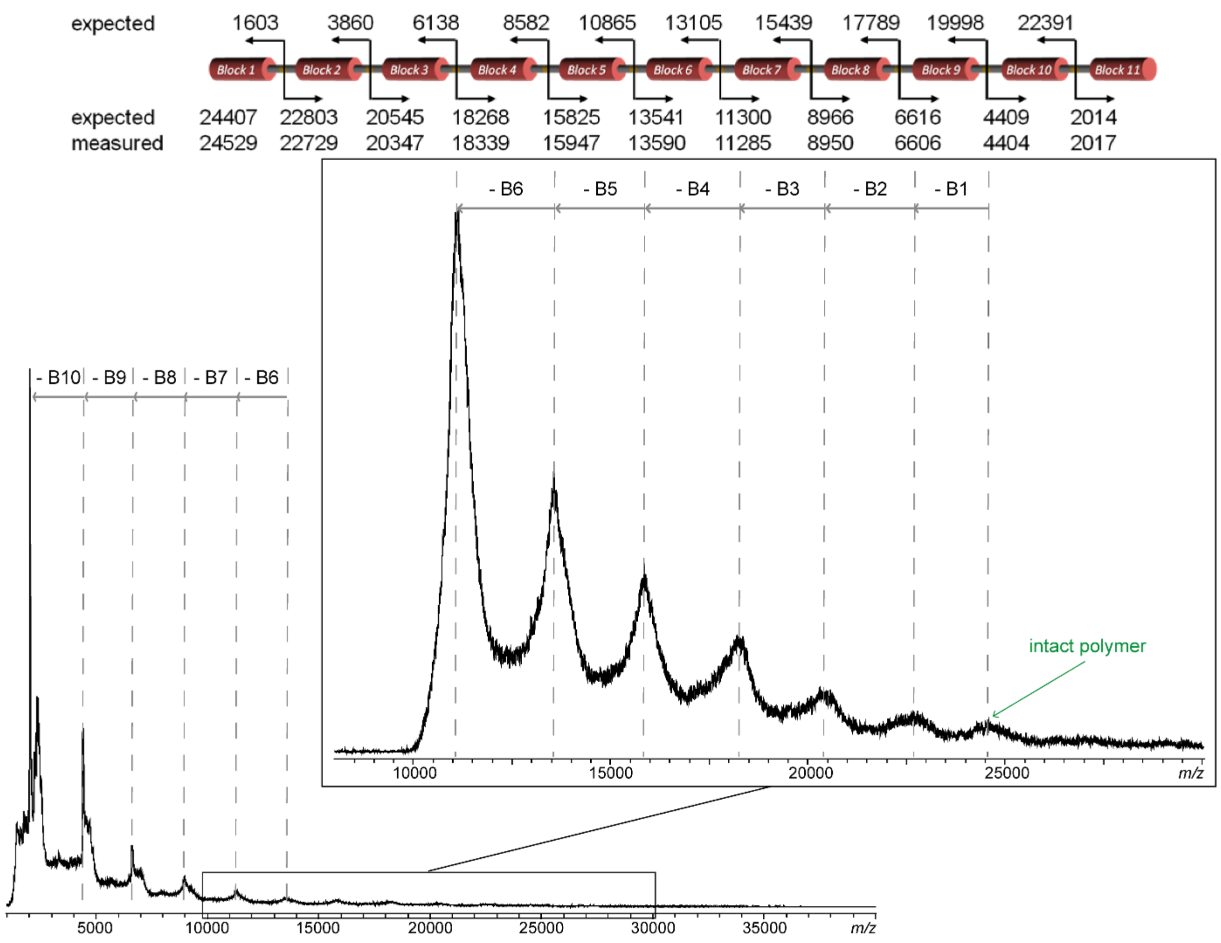


Figure S2. MALDI(–)-TOF mass spectrum recorded in the linear mode for the "Lavoisier" polymer. The poorly resolved signal centered at *m/z* 24529 was assigned to the intact polymer ionized as a deprotonated molecule. Mass analysis in this *m/z* range using linear TOF combined with multiple H/Na exchanges usually observed in poly(phosphodiester)s would account for the low accuracy of this measurement, as compared to the expected *m/z* 24407 value at isotopic maximum. Yet, this assignment was supported by detection of in-source fragments induced by the quite high laser fluence (55%) requested to desorb the polymer from the MALDI sample. These fragments are formed upon alkoxyamine bond cleavages that can be either competitive or consecutive. Based on major signals measured in the low *m/z* range, the preferential pathway annotated in this MALDI spectrum would consist of successive release of coded segments from the left- to the right-hand side of the chain (see *m/z* values in the bottom raw of the top scheme). Yet, due to large width of most peaks as well as additional low intensity signals next to the main low *m/z* ions, it cannot be excluded that release of coded segments also occurs i) from the right- to the left-hand side (see *m/z* values in the top raw of the top scheme) and ii) in a random manner (for example, alternatives to losing the three first (B1-B3) or the three last (B9-B11) blocks can be elimination of B1, B2 and B11 or elimination of B1, B10 and B11, leading to fragments that do no longer contain any of the original chain ends and expected at *m/z* 18529 and *m/z* 18393, respectively). Of note, defective molecules lacking entire block(s) would also contribute to these signals and their presence in the sample hence cannot be excluded.

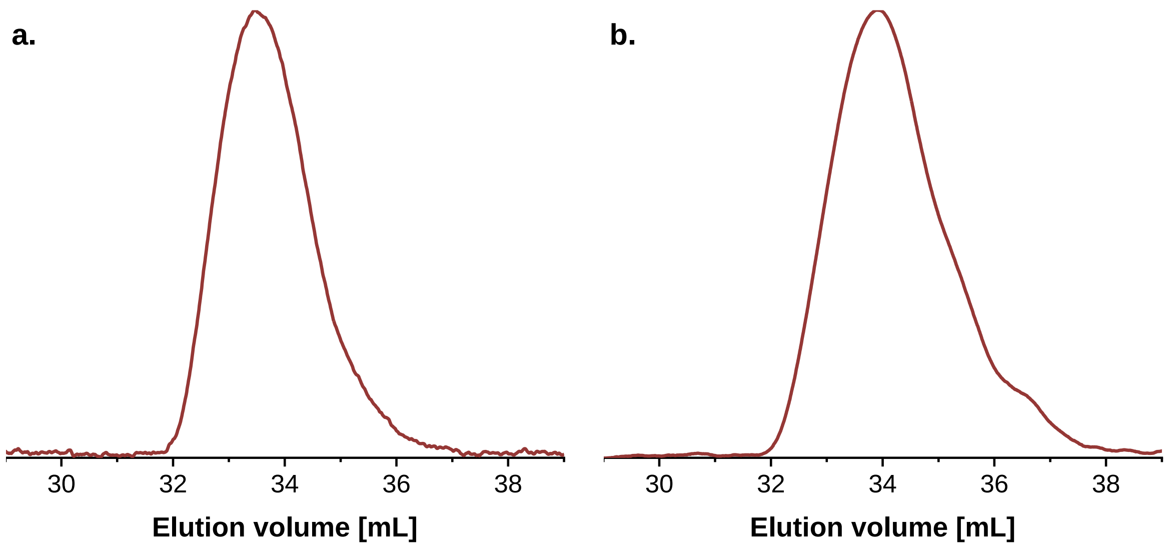


Figure S3. SEC chromatograms recorded in water/acetonitrile for the "Lavoisier" polymer: (**a**) Light scattering signal. (**b**) Refractomer signal.

	elemental composition	calculated *m/z*	measured *m/z*
$[B1 - 3H]^{3-}$	$C_{59}H_{116}O_{33}P_8^{3-}$	533.5105	533.5096
$[B2 - 3H]^{3-}$	$C_{85}H_{160}N_4O_{43}P_{10}F^{3-}$	751.2611	751.2592
$[B3 - 3H]^{3-}$	$C_{88}H_{164}N_5O_{42}P_{10}^{3-}$	757.6081	757.6054
$[B4 - 3H]^{3-}$	$C_{92}H_{172}N_6O_{43}P_{10}Br^{3-}$	812.6011	812.5952
$[B5 - 3H]^{3-}$	$C_{85}H_{158}N_6O_{43}P_{10}F^{3-}$	759.9246	759.9225
$[B6 - 3H]^{3-}$	$C_{76}H_{141}N_3O_{44}P_{10}I^{3-}$	745.5109	745.5085
$[B7 - 3H]^{3-}$	$C_{86}H_{161}N_3O_{44}P_{10}Br^{3-}$	776.2343	776.2321
$[B8 - 3H]^{3-}$	$C_{91}H_{171}N_6O_{43}P_{10}^{3-}$	781.9590	781.9570
$[B9 - 3H]^{3-}$	$C_{82}H_{153}N_6O_{42}P_{10}^{3-}$	734.5805	734.5792
$[B10 - 3H]^{3-}$	$C_{96}H_{183}N_4O_{43}P_{10}^{3-}$	796.6550	796.6528
$[B11 - 3H]^{3-}$	$C_{74}H_{146}N_3O_{41}P_9^{3-}$	670.5696	670.5679

Table S1. Accurate measurements (monoisotopic *m/z*) of individual blocks released as triply deprotonated species during in-source dissociation of the "Lavoisier" polymer (Figure 3).

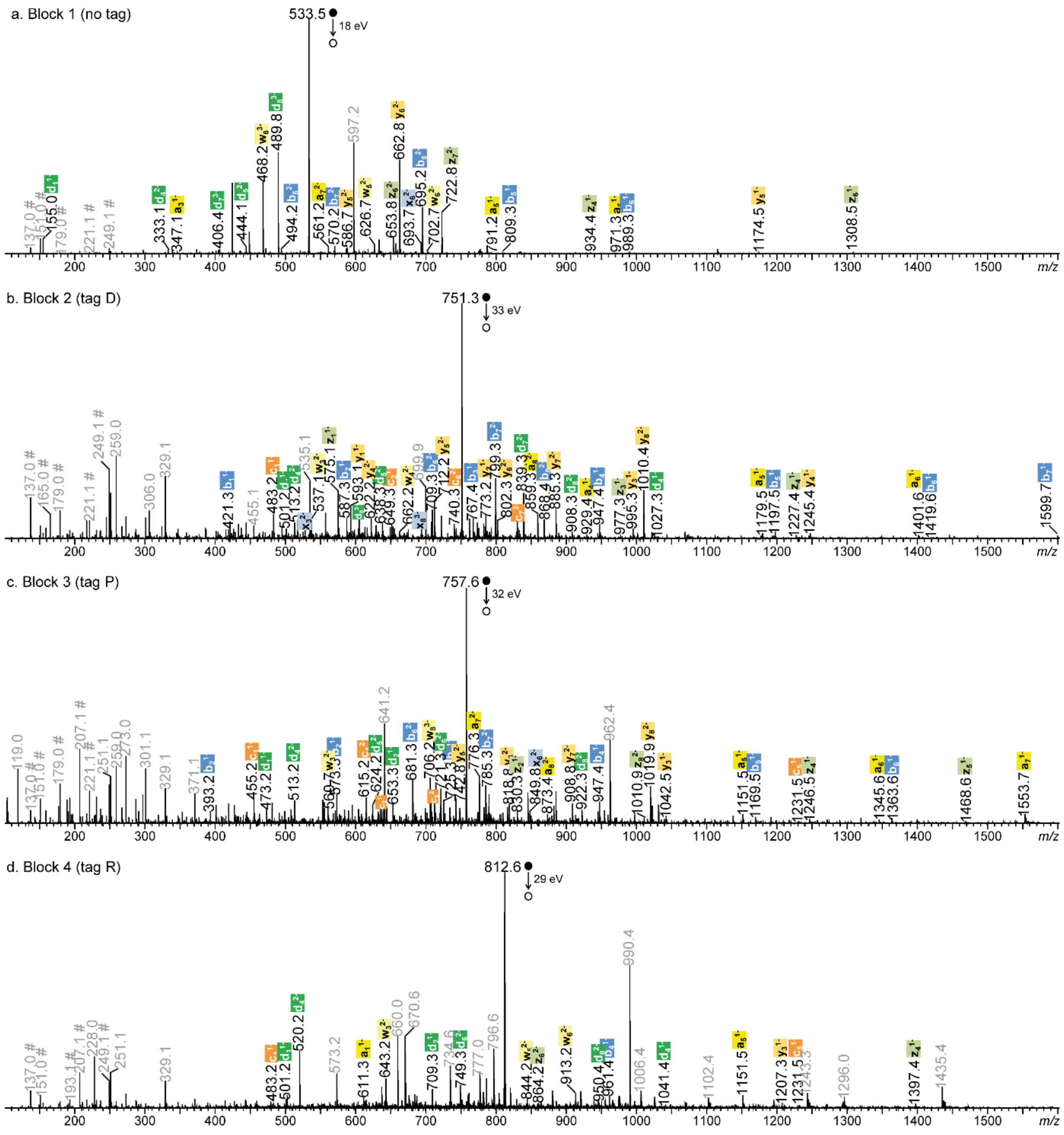


Figure S4. CID spectrum of (a) the *m/z* 533.5 ion containing the block 1 with no tag, (b) the *m/z* 751.3 ion containing block 2 tagged with D, (c) the *m/z* 757.6 ion containing block 3 tagged with P, and (d) the *m/z* 812.6 ion containing block 4 tagged with R. Dissociation of all precursors proceeds via cleavage of phosphate bonds in each repeating unit, yielding fragments containing either the α termination (a_i^{z-}, b_i^{z-}, c_i^{z-}, d_i^{z-}) or the ω termination (w_i^{z-}, x_i^{z-}, y_i^{z-}, z_i^{z-}) that enable full coverage of each block sequence, as reported in Figure S7. For the sake of clarity, all fragments were not annotated. Peaks labeled in grey correspond to secondary fragments, including deprotonated monomers designated by #.

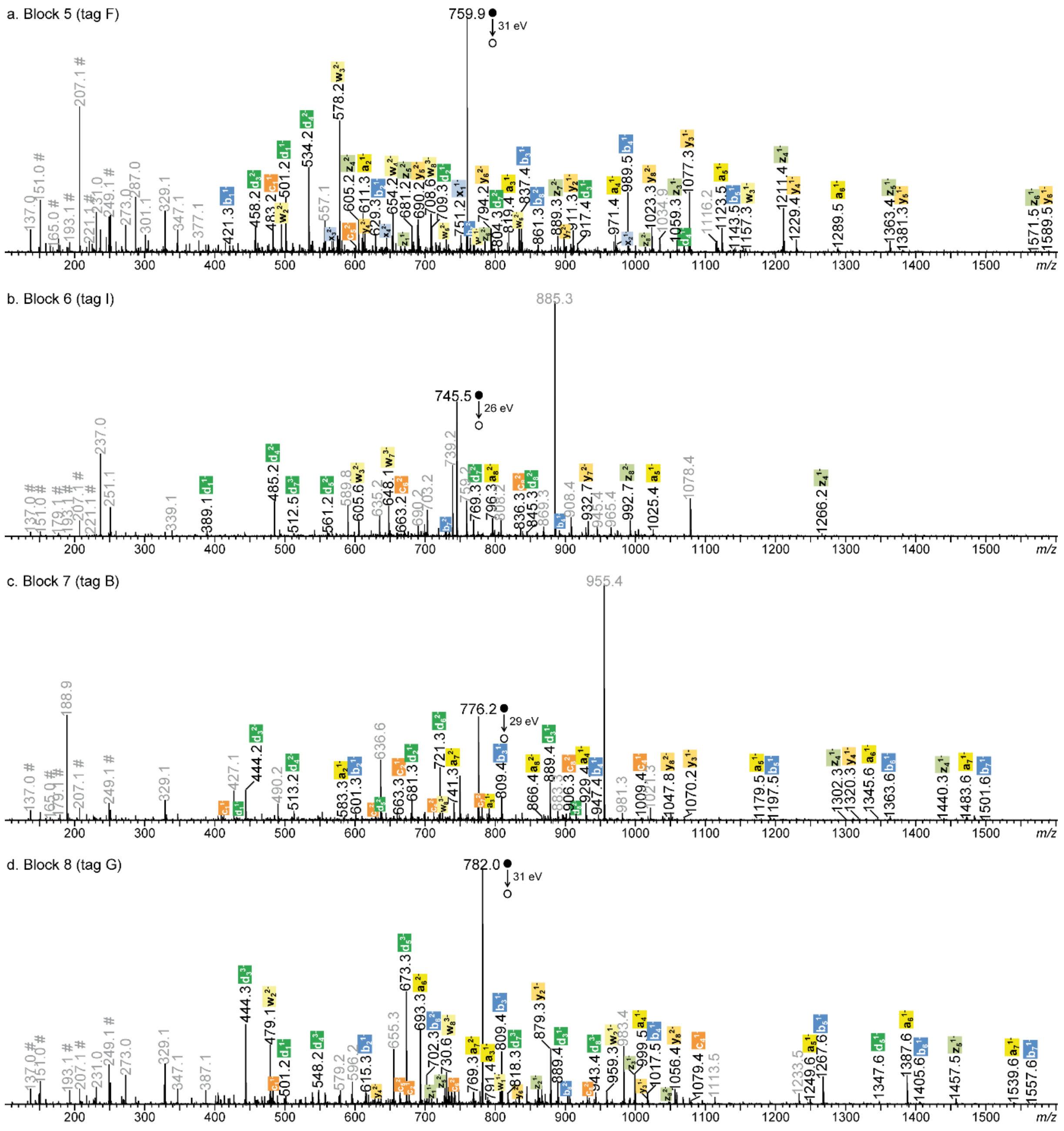


Figure S5. CID spectrum of (**a**) the *m/z* 759.9 ion containing the block 5 tagged with F, (**b**) the *m/z* 745.5 ion containing block 6 tagged with I, (**c**) the *m/z* 776.2 ion containing block 7 tagged with B, and (**d**) the *m/z* 782.0 ion containing block 8 tagged with G. Dissociation of all precursors proceeds via cleavage of phosphate bonds in each repeating unit, yielding fragments containing either the α termination (a_i^{z-}, b_i^{z-}, c_i^{z-}, d_i^{z-}) or the ω termination (w_i^{z-}, x_i^{z-}, y_i^{z-}, z_i^{z-}) that enable full coverage of each block sequence, as summarized in Figure S7. Peaks annotated in grey correspond to secondary fragments, including deprotonated monomers designated by #.

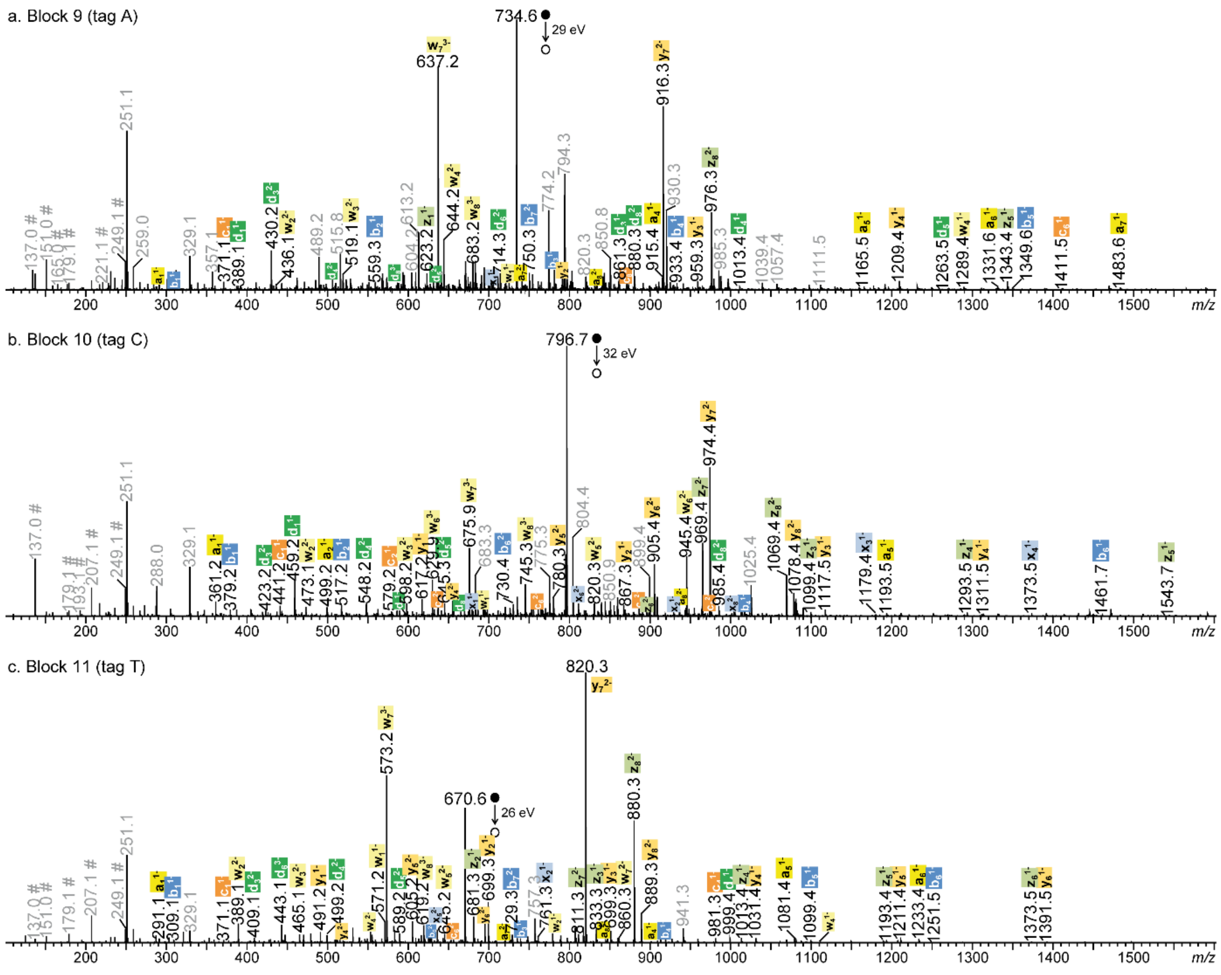


Figure S6. CID spectrum of (**a**) the *m/z* 734.6 ion containing the block 9 tagged with A, (**b**) the *m/z* 796.7 ion containing block 10 tagged with C, and (**c**) the *m/z* 670.6 ion containing block 11 tagged with T. Dissociation of all precursors proceeds via cleavage of phosphate bonds in each repeating unit, yielding fragments containing either the α termination (a_i^{z-}, b_i^{z-}, c_i^{z-}, d_i^{z-}) or the ω termination (w_i^{z-}, x_i^{z-}, y_i^{z-}, z_i^{z-}) that enable full coverage of each block sequence, as summarized in Figure S7. Peaks annotated in grey correspond to secondary fragments, including deprotonated monomers designated by #.

block #1 (no tag)

a_i^{z-}	*n.e.*	*n.e.*	-1	-1	-1	-1/-2	-1	-2	
b_i^{z-}	*n.e.*	-1	-1	-1	-1/-2	-1/-2	-2	-2	
c_i^{z-}	*n.e.*			-1	-2	-2	-2		
d_i^{z-}	-1	-1	-2	-2		-2	-2/-3	-3	
→	**000**	**000**	**001**	**110**	**110**	**011**	**001**	**111**	←
	n.e.	*n.e.*	-2/-3	-2	-2	-2	-1	-1	w_i^{z-}
	n.e.	-3	-2	-2	-2	-1	-1	-1	x_i^{z-}
	n.e.	-2	-1/-2	-1/-2	-1/-2	-1	-1	-1	y_i^{z-}
	n.e.	-2	-1/-2	-1/-2	-1	-1	-1	*n.e.*	z_i^{z-}

block #2 (tag D)

a_i^{z-}		-1		-1	-1	-1/-2	-2	-2	
b_i^{z-}	-1	-1	-1	-1/-2	-1/-2	-1/-2	-1/-2	-2	
c_i^{z-}	-1	-1	-1	-1/-2	-2	-2	-2	-2/-3	
d_i^{z-}	-1	-1	-1/-2	-1/-2	-2	-3	-2/-3	-2/-3	
→	**111**	**010**	**011**	**011**	**111**	**110**	**011**	**000**	←
	-3	-3	-2		-2	-2	-1/-2	-1/-2	w_i^{z-}
	-3	-2/-3	-2/-3	-2	-2	-2	-2	-1	x_i^{z-}
	-2	-2	-2	-2	-1/-2	-1/-2	-1	-1	y_i^{z-}
	-2	-2	-2	-1/-2	-1	-1		-1	z_i^{z-}

block #3 (tag P)

a_i^{z-}		-1	-1	-1	-1	-1/-2	-1/-2	-2	
b_i^{z-}	-1	-1	-1	-1/-2	-1/-2	-1/-2	-2	-2	
c_i^{z-}	-1		-1	-1/-2	-1/-2	-2	-2	-2/-3	
d_i^{z-}	-1	-1	-2	-1/-2	-2	-2	-2	-2/-3	
→	**110**	**011**	**001**	**110**	**110**	**100**	**101**	**100**	←
	-3	-3	-2	-2	-2	-2	-2	-1	w_i^{z-}
	-3	-3	-2	-2	-2	-2	-1	-1	x_i^{z-}
	-2	-2	-2	-2	-1/-2	-1	-1	-1	y_i^{z-}
	-2	-2	-2	-1/-2	-1	-1	-1	-1	z_i^{z-}

block #4 (tag R)

a_i^{z-}		-1			-1/-2	-2	-2	-2	
b_i^{z-}	-1	-1	-1	-1	-2	-2	-2	-2	
c_i^{z-}	-1	-1			-1/-2	-2	-2	-2/-3	
d_i^{z-}	-1	-1	-1	-1/-2	-2	-2	-3	-2/-3	
→	**111**	**101**	**000**	**100**	**101**	**111**	**101**	**100**	←
	-3	-2/-3	-2	-2	-2	-2	-2	-1	w_i^{z-}
	-3	-2/-3	-2	-2	-2				x_i^{z-}
			-2	-2	-2	-1	-1	-1	y_i^{z-}
		-2	-2	-2	-1/-2		-1		z_i^{z-}

block #5 (tag F)

a_i^{z-}		-1	-1	-1	-1	-1/-2	-2	-2	
b_i^{z-}	-1	-1	-1	-1	-1/-2	-2	-2	-2	
c_i^{z-}	-1	-1	-1	-1/-2	-2	-2	-2	-2	
d_i^{z-}	-1	-1/-2	-1/-2	-1/-2	-1/-2	-2	-2	-2	
→	**111**	**101**	**101**	**001**	**001**	**010**	**110**	**100**	←
	-3	-3	-2/-3	-2	-1/-2	-1/-2	-1/-2	-1/-2	w_i^{z-}
	-3	-2	-2	-2	-1/-2	-1/-2	-1/-2	-1	x_i^{z-}
	-2	-2	-1/-2	-1/-2	-1/-2	-1/-2	-1	-1	y_i^{z-}
	-2	-2	-1/-2	-1/-2	-1/-2	-1	-1	-1	z_i^{z-}

block #6 (tag I)

a_i^{z-}		-1	-1	-1	-1	-1	-2	-2	
b_i^{z-}	-1	-1	-1	-1	-1/-2	-1/-2	-1/-2	-2	
c_i^{z-}	-1	-1		-1	-1/-2	-2	-2	-2/-3	
d_i^{z-}	-1	-1	-2	-1/-2	-2	-2	-2/-3	-2/-3	
→	**000**	**101**	**100**	**011**	**001**	**110**	**100**	**001**	←
	-3	-2/-3	-2/-3	-1	-1	-1	-1		w_i^{z-}
	-3	-2/-3	-2	-2	-2				x_i^{z-}
	-2	-2	-2	-2	-2	-1	-1	-1	y_i^{z-}
	-2	-2	-2		-1			-1	z_i^{z-}

block #7 (tag B)

a_i^{z-}		-1	-1	-1	-1	-1/-2	-1/-2	-2	
b_i^{z-}	-1	-1	-1	-1/-2	-1/-2	-1/-2	-1	-2	
c_i^{z-}	-1	-1	-1	-1/-2	-2	-2	-2	-2/-3	
d_i^{z-}	-1	-1/-2	-1/-2	-2	-2	-2	-2	-2/-3	
→	**011**	**111**	**101**	**000**	**111**	**010**	**000**	**111**	←
	-3	-3	-2	-2	-2	-2		-1	w_i^{z-}
				-2	-2				x_i^{z-}
	-2	-2	-2	-2	-1/-2	-1	-1	-1	y_i^{z-}
	-2		-2	-1/-2	-1	-1	-1		z_i^{z-}

block #8 (tag G)

a_i^{z-}		-1	-1	-1	-1/-2	-1/-2	-1/-2	-2	
b_i^{z-}	-1	-1	-1	-1	-1/-2	-1/-2	-1/-2	-2	
c_i^{z-}	-1	-1	-1/-2	-1/-2	-1/-2	-1/-2	-2	-2/-3	
d_i^{z-}	-1	-1	-1/-2	-1/-2	-1/-2	-2	-2/-3	-2/-3	
→	**111**	**100**	**100**	**101**	**111**	**000**	**001**	**111**	←
	-3	-3	-2		-2	-1/-2	-1/-2	-1/-2	w_i^{z-}
	-3		-2	-2	-2	-1	-1	-1	x_i^{z-}
	-2	-2	-2	-2	-1/-2	-1/-2	-1	-1	y_i^{z-}
	-2	-2	-2	-1/-2	-1/-2	-1	-1	-1	z_i^{z-}

block #9 (tag A)

a_i^{z-}	-1	-1	-1	-1	-2	-1/-2	-1/-2	-1/-2	
b_i^{z-}	-1	-1	-1	-1/-2	-1	-1/-2	-1/-2	-1/-2	
c_i^{z-}	-1	-1		-1/-2	-1/-2	-1/-2	-2	-2/-3	
d_i^{z-}	-1	-1	-1/-2	-1/-2	-1/-2	-2	-2/-3	-2/-3	
→	**000**	**111**	**110**	**001**	**111**	**010**	**001**	**011**	←
	-3	-2/-3	-2/-3	-2	-1/-2	-1/-2	-1/-2	-1	w_i^{z-}
	-3	-2/-3	-2	-2		-1	-1	-1	x_i^{z-}
	-2	-2	-2	-1/-2	-1	-1	-1	-1	y_i^{z-}
	-2	-2	-1/-2	-1			-1	-1	z_i^{z-}

block 10 (tag C)

a_i^{z-}	-1	-1	-1	-1	-1	-1/-2	-1/-2	-2	
b_i^{z-}	-1	-1	-1	-1/-2	-2	-1/-2	-2		
c_i^{z-}	-1	-1		-1/-2	-2	-2	-2/-3	-2/-3	
d_i^{z-}	-1	-1	-1/-2	-1/-2	-2	-2	-2/-3	-2/-3	
→	**101**	**000**	**111**	**111**	**100**	**111**	**111**	**011**	←
	-3	-2/-3	-2/-3	-2/-3	-2	-1/-2	-1/-2	-1/-2	w_i^{z-}
	-3	-2/-3	-2/-3	-2	-1/-2	-1/-2	-1	-1	x_i^{z-}
	-2	-2	-2	-1/-2	-1/-2	-1/-2	-1	-1	y_i^{z-}
	-2	-2	-2	-1/-2	-1/-2	-1		-1	z_i^{z-}

block 11 (tag T)

a_i^{z-}	-1	-1	-1	-1	-1/-2	-1/-2	-1/-2	-2	
b_i^{z-}	-1	-1	-1	-1/-2	-2	-1/-2	-2	-2	
c_i^{z-}	-1	-1	-1/-2	-1/-2	-1/-2	-2	-2/-3	-2/-3	
d_i^{z-}		-1/-2	-1/-2	-1/-2	-1/-2	-2/-3	-3	-3	
→	**000**	**111**	**011**	**011**	**011**	**001**	**101**	**111**	←
	-3	-2/-3	-2/-3	-2	-1/-2	-1/-2	-1/-2	-1/-2	w_i^{z-}
	-3	-3	-2	-1/-2	-1/-2	-1/-2	-1/-2	-1	x_i^{z-}
	-2	-2	-1/-2	-1/-2	-1/-2	-1/-2	-1	-1	y_i^{z-}
	-2	-2	-1/-2	-1/-2	-1/-2	-1	-1	-1	z_i^{z-}

Figure S7. Sequence coverage tables with each cell reporting charge states of sequencing fragments measured in CID spectra shown in Figures S4-S6. Fragments containing either the α termination (a_i^{z-}, b_i^{z-}, c_i^{z-}, d_i^{z-}) allow each sequence to be re-constructed from the left- to the right-hand side (→) whereas fragments containing the ω termination (w_i^{z-}, x_i^{z-}, y_i^{z-}, z_i^{z-}) are used to read the sequence in the opposite direction (←). Due to the peculiar end-groups of the ion containing block 1, some fragments are not expected (*n.e.* in grey cells).

D. References

[1] E. Laurent, J.-A. Amalian, M. Parmentier, L. Oswald, A. Al Ouahabi, F. Dufour, K. Launay, J.-L. Clément, D. Gigmes, M.-A. Delsuc, L. Charles, J.-F. Lutz, *Macromolecules* **2020**, *53*, 4022-4029.

[2] K. Launay, J.-A. Amalian, E. Laurent, L. Oswald, A. Al Ouahabi, A. Burel, F. Dufour, C. Carapito, J.-L. Clément, J.-F. Lutz, L. Charles, D. Gigmes, *Angew. Chem., Int. Ed.* **2021**, *60*, 917-926.

[3] A. Al Ouahabi, M. Kotera, L. Charles, J.-F. Lutz, *ACS Macro Lett.* **2015**, *4*, 1077-1080.

[4] Y. Liu, X. Sun, B. Guo, *Rapid Commun. Mass Spectrom.* **2003**, *17*, 2354-2360.