# A basic ciphering algorithm using cyclotomic fields

## 1 Martin Rupp

SCIENTIFIC AND COMPUTER DEVELOPMENT (SCD)
18/June/2023

## Contents



### 1.1 Definition of the cipher

In that article, we study the transformation $\varphi(A) = xA$ in the cyclotomic *ring* $Z[\zeta]$ where $\zeta$ is an irreducible p-root of unity. If we consider a set of data as $(a_0, ..., a_{p-2})$ where the $a_i$'s coefficient are taken in $Z$, then that set of data can be represented as the coordinates of an element $A \in Z[\zeta]$. We will also be able to represent $B = \varphi(A) \in Z[\zeta]$ by the set of data $(b_0, ..., b_{p-2})$ where $b_0 = \varphi_0(a_0, ..., a_{p-2})$ , .. , $b_{p-1} = \varphi_{p-1}(a_0, ..., a_{p-2})$

There are no general rules to know if $x$ has an inverse in $Z[\zeta]$ or not, however there are always at least $p-1$ units in $Z[\zeta]$ (see : Computations of some cyclotomic units in Z[ζ] ).

For these units, $\varphi$ is invertible and from $B = \varphi(A)$ we can find $A$ by $A = \varphi^{-1}(B) = x^{-1}B$.

This raises the question to know if the transformation $\varphi$ could be used as a block for diffusion and/or confusion of data in the context of the design of an encryption scheme for example.

Block to be used for encryption:

$\ldots \to (a_0, ..., a_{p-2}) \to \varphi \to (b_0, ..., b_{p-2}) \to \ldots$
Block to be used for decryption:

$\ldots \leftarrow (b_0, ..., b_{p-2}) \leftarrow \varphi^{-1} \leftarrow (a_0, ..., a_{p-2}) \leftarrow \ldots$

However this only allows us to compute data with sets of relative integers, which are not bounded and cannot easily represent data encoded by digital computers. It seems a better choice to use $Z_r$ instead of $Z$, where r is any positive integer. Let's illustrate this with an example:

$p = 17, x = S(11, 17) = 1 + \zeta + \ldots + \zeta^{11}$.
In such case $x^{-1} = -\zeta - \zeta^3 - \zeta^5 - \zeta^8 - \zeta^{10} - \zeta^{13} - \zeta^{15}$

If $r = 3$ and $A = (1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2)$ then
$\varphi(A) = xA = (1 + \zeta + \ldots + \zeta^{11})(1 + 2\zeta + 2\zeta^{15})$

Note that we have of course $1 + \zeta + \ldots + \zeta^{15} + \zeta^{16} = 0$

A small program in C# (see [1]) can be used to compute Cyclotomic multiplication:

We find that $\varphi(A) = (1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 0, -2, -2, 0)$

The fact that there is some dependency between the input and the output appears to be quite obvious. However, we can form more complex units by taking a product of powers of the $p - 1$ units, $S(p, k) = \sum_{i=0}^{i=k} \zeta^i, 0 \leq k \leq p - 2$

$U_a = S(p, 1)^{a_1} S(p, 2)^{a_2} \ldots S(p, p - 2)^{a_{p-2}}$

Let $F_a(X) = U_a X + Y_a$ and the transformation $\phi$ defined by :

$\phi = X \rightarrow F_{a^0} \rightarrow \ldots F_{a^i} \rightarrow \ldots F_{a^N}$ where $a^i = (a_1^i, \ldots, a_{p-2}^i)$ is a set of $p - 2$ elements of $Z$,$i = 1, \ldots, N$ .

We name N , the "rounds" of the transformation.

We call K1 a ciphering key made of N vectors from $Z^{p-1}$:

$K_1 = K_{1,N}, \ldots, K_{1,N}$ , $K_{1,i} \in Z^{p-1} i = 1, \ldots, N$

And we call K2 a "salt" made of N vectors from $Z^{p-2}$ :

$K_2 = K_{2,1}, \ldots, K_{2,N}$ , $K_{2,i} \in Z^{p-2} i = 1, \ldots, N$

Then in the formula defining $U_a$, we put $a_j = K_{2,i}[j], j = 0, \ldots, p - 2$ where i is the i$^{\text{th}}$ round.
We also define $Y_a$ as $K_{1,i} \times F_i$ where $F_i$ is a fixed element of $Z^{p-1}$ for the i$^{\text{th}}$ round.

Our ciphering system $\phi$ will be the defined by:

- *the parameters p, N and* $F = (F_i)_{i=1,\ldots,N}$ ;
- *the key K1 ;*
- *the salt K2.*

The cipher will transform any element of $Z[\zeta]$ into another element of $Z[\zeta]$.

One of the characteristics of that cipher is that it will “expand" the plaintext as the rounds increase. This can be useful to confuse an attacker. Since the salt values can be chosen arbitrary big, the amount of combinations of the unit $U_a$ is theoretically unlimited: eg for salt values which can go up to M, any unit $U_a$ can take $(p-2)^M$ values. Note that the cipher could also build more units $U_a$ by considering the inverses units $S(p,k)^{-1}$. Of course a more detailed analysis would be needed to correctly assess the strength of the cipher.

## 1.2 Annex

### 1.2.1 Cipher example

K1=
120,355,365,513,483,397,736,785,926,945
K2=
139,635,627,281,452,436,213,279,77,77,764,438,829,363,4
56,182,167,791,474,448,782,66,81,388,314,529,96,112,465,
373,845,437,83,465,295,48,355,344,279,800,493,94,445,536
,977,278,183,714,463,70,418,670,935,942,254,43,415,389,8
49,301,482,917,361,974,655,173,773,88,183,955,437,500,77
1,344,856,668,508,754,361,40,632,904,853,898,148,949,731
,270,693,2,657,830,386,26,962,681,172,663,11,489,476,833
,998,965,32,136,490,241,853,485,673,673,688,480,136,725,
667,241,715,570,537,206,297,450,280,377,330,14,17,523,90
9,879,548,761,330,293,768,822,404,982,115,971,837,419,46
,134,123,883,72,314,822,71,322,44,588,555,797,853,788,31
2,223,810,478,347,881,946,312,496,297,540,942,409,717,98
1,202,559,890,900,780,146,583,716,338,743,820,757,630,82
9,274,674,983,720,262,271,688,498,872,735,414,637

X=
386,41,4,-18,922,221,894,917,968,769
Y=
50585423424676244242692871176812350802004159885793683739
92743452243311478745587401698474401409946924027633695540
49470425197553944524884976459703338429961163890374196390
82456003118317767386035505095616564536923819480803396544
07175968105632152790292384083721533254008092708855893231
36429710389428258103666943907875209260684486756296146943
11268974138561541003462629395607242142781927110084475413
95373698623717376859561684655521828268521758832481700507
81856649347534827814140717812266363906194545718982712008
54070402027676632635570087633698730042551179785875575090
68672795064542796019865894013393427150886473285946481484
61160980216492360680158284685063304681744894165410104272
82533061359098768431994516847606632419003776123590411750
04357441677105476386061315612272797688035333867590349957
29275393048244939755845108829071352076662697327769085009
98729189540028896514053967584141933150439305779312937803
17303930913516215284761929135611767004314474149929797479
10031038045417090869290166028860991159119929663896667205
77293947004149836609608938365820562729160045227391308301
72412655432624382087984265560151106409607514715855993762
43507004017069258642206541458791825835850365062551744809
65661683064448553728207895413880291669443634982967857481
19823619933347634725698112294218077682319909687776572171
57454558644295549005792194948045396302413170523862437626
19790744362013275829981479732724136816297239768390745244
84278523297990981436672241188869658322243809594088392077
72493036343011853889221544491182635905755617782022189371

336728068293024626983819471120844862038565449114470300674896645399315310957603303302473572134547978930180905228354601513770849758429457146216606627018631200362594631967359413230410299823085291496174892766521453751557184611396447024905187072081058462809478802894471989087808565322589983892751842854840945115986940954105361293887328243937785515376827267618645373485337979822625025709589283328511085810444085872027515607828001453686330303211670321410743017560819702356260,964176153470839085924293046865024062713770259344243492261616030814122709264563981286626534994207637652103842086083337778930621286997719367098170215644890138555902382701338059211033515781432892482412765573496927252677352427654687501132883945701323533966894661001045360153133816049544791167964246671121916201517469290992493543168541635243776934470933086933016409238421404906690812212531229841580886001489330649135706305856260718990401277800314088261826938810496997904609796600839248448923806094498327887756082387475705299552525172249224260553310204283425581110347989660636170566571827481253594117177730111060232892348305829923837213140720092156744990670529536244912945860799791890850231554788418567706112468858094253167868129103108833074612511678064695703105453440918867034781496884950272256362231470168535454109853302787019279471407384617569414178318449360907325728193131753201247955933638168294425102675934684665306363241576493106959276181197000630907191386920750657220449000932505005269957323088598911546098146522676615273562732696361623892595644961071004456659451483141061747481142397377535140966292397749001072494803643940256191927963617650596745947081364575751901676180019043431160187178608064029329259119067107733255595138991117140159240311525688873103603830125212894564345993466580273523586412054073027120585140777473662744405843115180034825024886928486031114533926914537640722882191316549968773293424352726364285157893725500958580007848177515461733523254746024188487149218974274780963592669522406785116977915323751736755763673895194086861959560762974013312639369481395443126305996896029374718107571085931370182605905697477153314818380094384080669159264753251876935613327978872433054808586903749950311740085855081651616840030476808351774713026418173392410926565950097944014729459280139348327524107650102934527148061260168338296186236642307099815590289872378832080079126768663119332471477795997536914092490697226388775050612070920431223432653901500297218381428939332234310223921,122945178666652009213040318869119814875446187961594003734371731167369204057174007011377221509831823788406588645019426581863183392675699616592460457372460115116985480280367966355153237794389146366907000016903190850126055544623843854459402487757638763902418072306063416221039430526496929444639375224197572565282255893790423266864493277612

743483169117874700464706089630351002145585629968975760729651233002161789802694112792162935534872558379268951974405976172014606123215939900765231500129809105737914024880019288093533694233575480455652450829562372309905800075695478331774555528943899182367128160661731133418744295902507376625360069257679228904392900442540232242007231782347869751111391046788519741666848134510610359274267704323467054614417963361333063042230002992334141607973544916942016646481421883198190955624766290821698875787944971627077662709920697965832111080246564637661216858811927211704601578974620127044688451808979348333427461327655645599925946679058258918979179905973416265322749748808858585346032827496082162696395283400567679149138061705709658462368980121645773161450250219814715666302672936486272383839683151417849474296517387115353112590658573485143575685226495463326879338063289106009562539952870948540541454540201585239511033283234765988273472589215011766924949184992794381453961990756386942225168496849829737834119585892350791827193065553769932097515148194560738474359393938867856970679051196941555762071804246945503172549919715062224096439007980341885648577271794482752592731632925350329992802806818180882300395721324436006937580823043240281046002008411859392257374211846255114749703566356896092144248095937816521774055106324141716725898234240292058764903982933687011098449137618793987474466555005905137388928976500186949529264146902240494173522456113146666522903530953272009672217470989390783309387720095151032393556615238895856724996431228430961489049037934679836199079565077074583899366113331014564379445975474132058028629376043552176205136740423623657438524279,12174579946422338576057336914914955224449760462272463617744799707288767382321039968574797956441592840086326130077250991293091338618914402207374747364675771793256374433573227081104428207994678121307413639925060919350767390681891439204971117107804322466307254150660774832872098790457860820909592991359255562800870755970200030425485402309467534944705613937330464410330731933715797941244037140006746404850250990134848729453768823764712433603026839115167254757885663396833329571953153554446298241126569717842974914829855688273895214542378387169198508544989974704558990655744464091416370717308539031313279701774521556853761744812226636897427607405923469633988093661910811072339987331885247426886958751840348138147886642047658581084305395322299369642844175949923459429840438144341170380728801327549690932830813818972279037740844980689985692044303305716292622068190656572134909387074804500582916965285449119615615627936533478018030385276595957265262638572931436034020353109499080369703755045220554361501471857996439783524261346767745093508591775491574805686209563508976839944081223170462274155228369147419793949705935660505937012449257266780261400398789266058766402656783761773988172

012726658209099823786435839216532766766058320900698828423929647625135864702271494563281765797036799428941789932465688572006097301879683748106749741631616004135781444363889807422120888381493011452044038102923947343008645117334235480147591758119923916571934256176391736479341212847212047445037104700573483703334702811858433830102555428340445487487017546981606929243475965553545496628383749072559653877098570234304613211925719794290897836961125604798348576389734375747252478624851607631168339410007132401290891986398081309883128750887272581906612559908668098393428137787720185013535661305765939868564395699360496664794681169154910484325499972989314133444041294881216855362665782767029959241965482573742828446078806188567238591404153291750872503771477154539163347868175937346018880813394702097409095556775751,9320027216216063741062430254419279519521449776936511540829442292300650113785131254539344152821731776750607141840781731025732714599899763951059169444323013441488395998552347328456711392261595730479258330148007765849731068445589116203559641835304750486157828026651957404784132364978924323858870866663641640292127905477381798168419579325037827608089757040864537369822438237968081106346447646038192735874182118079331471995770683170696045635699819918372357665971194743555443481395677576072390145016491021560330854079194704653888242028938494847963506879105685097954449188156790256300586286633925955768659976951800619689298704797406050143680917424341342709121290828874128913005804268785312971392677384555105384208288187030728350037941001434999645938960386122456505623369927066906141936707983393137423447608442061868814644330080457274213134123119597399988883586208530854712972851167634694160850864901134212248041769570876862192743400432056294793188792604412343529703934303600995277575937999801577320042354696920887051396898551172150678765982398110600014721886595065546242000049160026685396393533515263446030500219389098377266585335868221139730871850415415289927318188683691826106559185238322863703801293271697608187517461815749608006115904206772969401067031685006050348076643444561660985727234136813257441166716020950454692860747834223527441296963641355336896767078504542071340564961051057156781629904749377625480418120448105951533902873511041152423649319324032195038170342045955359023119946306175745395914570918800420589838656515973342430679361295059695137214218398788274436339975995835651353147015914133411368769529685900843722661793789691624696583285762250192250129647332437669550715168686959924091066886236570039131764247762534160670728843075986371379563361128096351850091941988337347347781507095042015329139768952060836885501633472299734557379634533656959240022795512638421083243366334048929145645018973094252316220495522753630201292413192130380016554262725580808028526012210934788258,

463715999858108154956370597885882863621975755127091386797
764359908715469099770208175677124328766972740435919347247
926324231096575578775519580668902665132376212411476918307
645318823423296054322891189496124798145003340890011796967
350093536035851373049055853859645835130617492952219574387
995480748746427142847013425923824541453249649940498806927
532300039579980017080062586819996658881102355521303130827
662057149281309623646559336610966065769017824526324261077
340914903822283403459647330178690821466170501821271195867
053234596429744323040268338962493022069586209383431765277
420467219052446773227180150733995608979777687495800506257
701980303691317482519863576454450955700136956045807272807
659563276284214447015213125757015203529195917472733124347
588380229852967755987399667270011985433515292028614217697
469743489641149500066096163579386177022996925312118106237
420618252826173950013491200970829902636276010942077461637
615736812785437939157950700452975799514022234727031680437
984822415548303405221467900422414868303519504102623511497
143035840466478030637883527567636516046561757645284820787
734833145185219234079388352817794453372418750364408157937
557201581591135531530151041811384750308633518861970294037
142920933850547700767772760565816660105001648224220153687
744039559096044254382492367261934040949599086766471172887
296050586617667246603385716153611222546891402031357990737
371079136630585287688761200468824395553707522709948399917
977199449765933378059851128007481650000174259712255298477
141532904290665837782926606479350312193517448917463852087
571087756786306227427481467414408322546673305324019339967
482986514386409566142678769987341986187292504588722920977
852579741409646778814784559902396996381344848102743051737
372816877591147808935844999061238364377808282656628350577
917186079478254188980553983737103791235974877939286540237
611487502998691104380929847305218597678200293758609487367
164589811387325908508000410910271203299067302287523396767
749419805284532288852255887327640337437528395780070773817
32868958656352517838,
-387244452442148621309200129443366677174774392735428813647
073551232741336204387483221006324971069985393359578650187
876227547948232675174461851417863581930411288304906268387
454251147439497396097366310061489655055825484216413797567
843456041349165797786841345588048611229922085468527939767
244122636630775534248385577929052742856885248854936984697
810744793657690440075511830988794543006088642265160967577
235564562559701853871087159697292120088400026025563117867
956929684623912936451255057160020561288553365331012255327
733437829605070363184629940411970803684991130328003714077
250076794043713715485310098998307384895940943799023634987
864238265139626071830526026915412863196205851772464621687
521742176613381633899038490377934374819116477156978439557
738401571919689749044691512290005460505086899772000072147
195128924064443353184769497191429992321089449103659401497

16131703450270896678506002063234807647199241885583474731
32429045368014952933041364312279950082223311205219758844
83575025986741343486378208939821448376555664618600387822
57361402207395719645214564805408618930660471595620477174
95144089121238956327322867244806713497242047155899260644
67058051126374913296952267623369661279118451958241661246
66442526257922211203742716337951508424848920941183541185
22893939104067850569773756567351907692538452933446696452
89188321121577664625627641474320960282683359993162540715
03916609861444227224983818514413784280330859794028038954
12876001332667020452991204920218915661655152955039215037
98862712144265150437685803161083951560032642713487111259
61674203946081986154754862381787847091138858379269982331
80260627289891299900221563735448655562421430372545318989
59891311763560044386933344389471291242165104199284283687
02843043111378214759705188798108207928049541181612199416
07429537567724489737885326556830662632858877000827749353
10266751420921916356005237979101579148595113881887176010
25783922221932061559424661490207556927029637302854895343
57079766585176755385054508069194463603662538223723126318
0257543727458230066,
-41579732243991739492740224939763528763612931002311613178
78268615810956667868299834485346449803144035860258447023
30258536840155236332142797461199912277914900315377236051
43220046211973556053099852033807645793783470969700946591
77938175690792181796517190776637981372565559945846881137
17561476715732213151233978555256621691347269331462693857
33853868226422960928881746172729648099102265028958544840
24680237484072829745403549898119281733535444899535231220
64391724029051068939829784537559333194454224107028816230
43167572340880008402525104836133814951840829376776747719
39446922292474750389404535503326105293025528575406969576
30151746378455920061581241080981389190791987327109246330
08187831573670069779565945489389914288309491336379674368
60416900815237482425256593682545341615304348276258260590
50514974809346004800854962744684983271572972165406200199
34065021209694856325590329122300922605353406167004469165
29195955513104789975853808848679262979442956503584959239
76522858222546315766263831659146087685034120817982736009
85595525614198347517717858070235929879819435262909052565
05199504322752081093278618232195749329926829535900257936
86178744361315194857362432717628523476069203381288528939
21593147200755565918519067343914143039708224823360207169
26505046332478974239637573828367437267746017151427639619
99404173648174204220953802465684886798952067364222203885
14855082492252527458165018570795653639789493426785467030
79494529468327057652669763402451030802605891634722000759
46269203969587933737742217078817548552628859653149333247
36247011614296835559700940086503808250543841723893251800
05687322090445065907963256676070573025248073341959878514
69700102790623684041104409447914236929441286261866760734

02946053739329813483579996729917350491840130778727939972
04957025368658285444862177219196815921980364413046355877
65118903332702175693571784007680708997488413803696132115
56543047430160509583841547762345987340512644793902511933
10471333875071431536661124650677948466845355485090281088
28807304413492095555,
-54778465748900014934010546344888311108565580985133064900
43351557421545115741657013508210029083115213110577222345
89350615815725320029858818203543318672967780596417387363
72857478383972447131518526194929033936008469675193731374
86944450763567039481976640728448931921058639508636905790
43180209213950201433786440190265162967330612255139549037
83981846232641458710114574238314366082496744355937231153
90615162237597808169359739325657369442153352247109099046
68798037494217704302935063133116529229051496039411741591
62104430331804709049867712684480894624076165851951184768
38133590045725478751669198890905223746977937208846134943
94956836087375340833038170237052844052945512652045042337
65293371485151069734049409125031603456068067807244109539
36107962560496700058944223336856086083585746792502589315
47545830598407325519714573880203933432491338266893310001
67086940204012688741159160796615544368575062197737467246
77213313492399060564325922173395751852438039686015695955
23242601241110286046252873956550054376886278589630819055
73808185959508575767672571133043234282411338484018236948
12361657912086261991449290714018742523639638912815701094
27489146627560716164481923959215416865184100154242238527
85194405366549303409680333786545558085833325334022041444
09961044607580639847069891558502027391344048454959671320
90234236586989143900159217727694666180033258366100579490
96263787204903098791883874156146943921835222416163521347
80730789719053199901227522405069111193900285046738852198
08366802498757371475277901220964348090273597058626636253
19829159404917832988690681107276315774398760500891899494
73262470307588326024137954301955971402253416819880935226
59836516577562730665222625048189445108927113011763019183
13048841031688990283477118011042562119101803936016377053
92023633858751177301485934609235604293513604316929230113
02130608541525026668484976925757646848399737988309571480
52825747051665329973792497701054446283864797565315808233
18154274415080198371256937938425847636977365975071473300
19315599244568697996,
-39278140409132491240567788396537476398745434376658073872
20592123358749631262226418494915229549817202788865817578
93278388333256772133045958893197097502640332661901902317
25645622158546692895320561906524721790910063228173265405
29347671142940127610970812024904268018011160922248857261
86421026216007192334217521267918279352199192485696533882
16468249312393988274412954304817759026911974048828964335
70825430132564132211947385317350601025205868458316148681
50268918805160809968230594747655583734704670315727714289

122546214813072877838770696094380144883940884359646769429689441932627377538170142293154631020061680867885497968224311685094628113807649625334387859953003240328025193918399870813391716172544959127290376762528114830182329253363309766785818931931406107690109253556176504373561616771267796698402999795315865018953133883537457330728626792101799664577672464551850953266660671385301766495710270814438273590241387262659614033198559060031028600886801341498851185421349009962082733689627841534028041624690945407407966744026997144393818966195333441855257975127486105189463708444187848817526218608836052626586069838837942521866745729419139116574271166523142155591093843235052130473568855384668556142856357254504286440919467400713798244232344948567778772689743590273793122684900291559468732810433104868584176242213666796761199676821690017143822707149700872823997693826238180457955830467601635189116708267618351092803465749421843243806052044524544132391085867303952491157640073468175519850491832544014544498178173045585313770235762465282501925428983428851471523647049274635345408439515834745845370028987460646206131309655621700269980168135551672210224507291010779126693952332738739982955535868444166236572273119159330120030150549379177791050593896807480173896177046330098424054372016592648560351894718565636042387724199005361273021259464936237987410496984067616362677271225428410264195553684556379000668811669508903108034908622366153087279180764131158406271466364640462161332231655940972
X2=
386,41,4,-18,922,221,894,917,968,769

### 1.2.2 Sample code (C)

```
using Cyclotomic1;
using System;
using System.Collections.Generic;
using System.Numerics;
using System.Security.Cryptography.X509Certificates;

public class Program
{
  public static int mod(int x, int m)
   {
      return (x % m + m) % m;
   }

   public static BigInteger [] Add(BigInteger [] A, BigInteger [] B, int p)
   {
```

```
    if (p < 1)
        return null;

    if ((A == null) || (A.Length != p - 1))
        return null;

    if ((B == null) || (B.Length != p - 1))
        return null;

    BigInteger [] C = new BigInteger [p - 1];



    //start computation
    for (int i = 0; i < p - 1; i++)
    {
        C[i] = A[i] + B[i];

    }

    return C;

}

public static BigInteger [] Exponent(BigInteger [] A, BigInteger N , int p)
{

    if ((p < 1)||(N<1))
        return null;

    if ((A == null) || (A.Length != p - 1))
        return null;

    BigInteger [] C = new BigInteger [p - 1];
    A.CopyTo(C, 0);

    for (int i = 0; i < N-1; i++)
    {
        C = Multiply(C, A, p);
    }

    return C;
}
public static BigInteger [] ScalarMultiply(BigInteger [] A, BigInteger m, int p)
{


    if (p < 1)
        return null;

    if ((A == null) || (A.Length != p - 1))
```

```
        return null;

    BigInteger [] C = new BigInteger [p - 1];
    A.CopyTo(C, 0);

    for (int i = 0; i < p-1; i++)
    {
        C[i] = m * C[i];
    }

    return C;


}


    public static BigInteger [] Multiply(BigInteger [] A, BigInteger [] B, int p)
{

    if (p < 1)
        return null;

    if ((A==null)||(A.Length!=p-1))
            return null;

    if ((B == null) || (B.Length != p - 1))
        return null;

    BigInteger [] C = new BigInteger [p - 1];



    //start computation
    for (int i = 0; i < p - 1; i++)
    {
        for (int j = 0; j < p - 1; j++)
        {
            int k = i + j;
            BigInteger c = A[i] * B[j];

           int   k_ = mod(k , p);



            if (k_ < p - 1)
            {
                C[k_] += c;
            }
            else
            {
                for (int m = 0; m < p - 1; m++)
```

```
            {
                C[m] += -c;
            }

        }

    }
}

//end computation

return C;


        }

public static BigInteger[] InverseUnit(int k, int p)
{

    BigInteger[] V = new BigInteger[p - 1];

    Array.Copy(Enumerable.Repeat<BigInteger>((int)0, p - 1).ToArray(), V, p - 1);


    int s = 0;
    int t = 0;
    //1) Compute s using the extended Euclidean algorithm
    Euclid.gcdExtended(k+1, p, out s, out t);

    //2) Compute the residues of i(k + 1) modulo p for i = 1 . . . s − 1 if s > 0
    // and for i = s. . . − 1 if s < 0

    if(s>0)
    {
        for(int i = 0; i < s; i++)
        {
            int index = mod((i * (k + 1)) , p);
            V[index] = 1;
        }
    }
    else
    {
        for (int i = s; i < 0; i++)
        {
            int index = mod((i * (k + 1)) , p);
            V[index] = -1;
        }
    }

    return V;
```

```
}

public static BigInteger[] Multiply2(BigInteger[] A, BigInteger[] B, int p)
{



    if ((A == null) || (A.Length != p - 1))
        return null;

    if ((B == null) || (B.Length != p - 1))
        return null;

    BigInteger[] C = new BigInteger[p - 1];



    //start computation
    for (int i = 0; i < p - 1; i++)
    {
        C[i] = A[i] * B[i];
    }

    //end computation

    return C;


}
public static Func<BigInteger[], BigInteger[], BigInteger[], BigInteger[]> CreateCipher(int N,
BigInteger[] F, int p)
{
    Func<BigInteger[], BigInteger[], BigInteger[], BigInteger[]> f = (K1,K2,X)=>
    {

        //we must build the p-2 units
        BigInteger[][] U = new BigInteger[p-1][];

        for(int i=1;i<p-1;i++)
        {

            U[i] = new BigInteger[p-1];

            Array.Copy(Enumerable.Repeat<BigInteger>((int)0, p - 1).ToArray(), U[i],p-1);

            Array.Copy(Enumerable.Repeat<BigInteger>((int)1, i+1).ToArray(), U[i], i+1);
        }

        BigInteger[] Y = new BigInteger[p - 1];
```

```
            Array.Copy(X, Y, p - 1);


            for(int i=0;i<N;i++)
            {
                //N rounds

                BigInteger[] U0= new BigInteger[p-1];

                //we form a new unit:
                for (int a = 1; a < p-1; a++)
                {
                    if (a == 1)
                    {
                        U0 = Exponent(U[a], K2[(p-1)*i+a], p);
                    }
                    else
                    {
                        U0 = Multiply(U0, Exponent(U[a], K2[(p-1)*i+a], p), p);
                    }
                }

                Y = Add(Multiply(U0, X, p), Multiply(F, K1, p), p);

                }


            return Y;
        };

        return f;

    }

    public static Func<BigInteger[], BigInteger[], BigInteger[], BigInteger[]> CreateDecipher(int
N, BigInteger[] F, int p)
    {
        Func<BigInteger[], BigInteger[], BigInteger[], BigInteger[]> f = (K1, K2, X) =>
        {

            //we must build the p-2 units inverses
            BigInteger[][] W = new BigInteger[p - 1][];

            for (int i = 1; i < p - 1; i++)
            {
                W[i] = InverseUnit(i, p);

            }

            BigInteger[] Y = new BigInteger[p - 1];
```

```
        Array.Copy(X, Y, p - 1);


        for (int i = 0; i < N; i++)
        {
            //N rounds

            BigInteger[] U0 = new BigInteger[p - 1];

            //we form a new inverse unit:
            for (int a = 1; a < p - 1; a++)
            {
                if (a == 1)
                {
                    U0 = Exponent(W[a], K2[(p - 1) * i + a], p);
                }
                else
                {
                    U0 = Multiply(U0, Exponent(W[a], K2[(p - 1) * i + a], p), p);
                }
            }

            Y = Multiply(Add(X, ScalarMultiply(Multiply(F, K1, p),-1,p), p), U0, p);

        }


        return Y;
    };

    return f;

}


public static void Main()
{

   // Euclid.Test();
   // return;

    System.String input = "";
    Random r = new Random();

    int p = 11;
    int N = 20;
    BigInteger[] F = new BigInteger[p - 1];
    for (int i = 0; i < p - 1; i++) { F[i] = r.Next(-100, 1000); }

    var f = CreateCipher(N, F, p);
```

```
    BigInteger[] K1 = new BigInteger[p - 1];
    for (int i = 0; i <p - 1 ; i++) { K1[i] = r.Next(-100, 1000); }
    BigInteger[] K2 = new BigInteger[N*(p-1)];
    for (int i = 0; i < N*(p - 1); i++) { K2[i] = r.Next(1, 1000); }

    Console.Out.Write("K1=\n");
    Array.ForEach(K1, x => Console.Out.Write("," + x));
    Console.Out.Write("\n");

    Console.Out.Write("K2=\n");
    Array.ForEach(K2, x => Console.Out.Write("," + x));
    Console.Out.Write("\n");

    BigInteger[] X = new BigInteger[p - 1];
    for (int i = 0; i < p - 1; i++) { X[i] = r.Next(-100, 1000); }

    Console.Out.Write("X=\n");
    Array.ForEach(X, x => Console.Out.Write("," + x));
    Console.Out.Write("\n");

    BigInteger[] Y=f(K1, K2, X);

    Console.Out.Write("Y=\n");
    Array.ForEach(Y, x => Console.Out.Write("," + x));
    Console.Out.Write("\n");

    var g = CreateDecipher(N, F, p);

    BigInteger[] X2 = g(K1, K2, Y);

    Console.Out.Write("X2=\n");
    Array.ForEach(X2, x => Console.Out.Write("," + x));
    Console.Out.Write("\n");

}

    public static void Test()
    {
    System.String input = "";
    Random r = new Random();


    try
    {




        int p = 97;

        BigInteger[] S1 = new BigInteger[p - 1];
```

```
        BigInteger[] S2 = new BigInteger[p - 1];

        BigInteger[] K = new BigInteger[p - 1];
        int[] K2 = new int[1000];


        for (int i = 0; i < p - 1; i++) { S1[i] = r.Next(-100, 1000); }
        for (int i = 0; i < p - 1; i++) { S2[i] = r.Next(-100, 1000); }
        for (int i = 0; i < p - 1; i++) { K[i] = r.Next(-100, 1000); }
        for (int i = 0; i < 1000; i++) { K2[i] = r.Next(1, 1000); }

           Console.Out.Write("S1=\n");
        Array.ForEach(S1, x => Console.Out.Write(","+x));
        Console.Out.Write("\n");
        Console.Out.Write("S2=\n");
        Array.ForEach(S2, x => Console.Out.Write("," + x));
        Console.Out.Write("\n");
        Console.Out.Write("K=\n");
        Array.ForEach(K, x => Console.Out.Write("," + x));
        Console.Out.Write("\n");
        Array.ForEach(K2, x => Console.Out.Write("," + x));
        Console.Out.Write("\n");

        BigInteger[] A = new BigInteger[p - 1];
    BigInteger[] x1 = { 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 };
    BigInteger[] x2 = { 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 };
    BigInteger[] x3 = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 };

    BigInteger[] x = new BigInteger[p - 1];
    BigInteger[] y = new BigInteger[p - 1];
    BigInteger[] z = new BigInteger[p - 1];

    Array.Copy(x1, x, x1.Length);
    Array.Copy(x2, y, x2.Length);
    Array.Copy(x3, z, x3.Length);

    BigInteger b = r.Next(-10, 10);

    //input vector
    for (int i = 0; i < 16; i++)
    {
        A[i] = b;
    }

    //BigInteger [] C=Multiply(A, Multiply(Exponent(x, 2, p),Exponent(y, 3, p), p),p);
    BigInteger[] E = Multiply(Exponent(z, K2[0], p), Multiply(Exponent(x, K2[1], p), Exponent(y, K2[3], p), p), p);

    BigInteger[] C = Add(Multiply(E, A, p),Multiply2(S1,K,p),p);
```

```
BigInteger[] F = Multiply(Exponent(z, K2[4], p), Multiply(Exponent(x, K2[5], p), Exponent(y, K2[6], p), p), p);

BigInteger[] G = Add(Multiply(F, C, p), Multiply2(S2, K, p), p);

    String result = "(";

for (int i = 0; i < p - 1; i++)
{
    result += "" + G[i];

    if (i < p - 2)
        result += ",";
}

result += ")";

Console.Out.WriteLine("result:\r\n" + result);

    BigInteger[] H = G.Distinct().ToArray();

    Console.Out.WriteLine("distinct values=" +H.Length + "/" + (p - 1));


    //+2*(1,2,1,3,4,0,0,0,0,0,0,0,0,0,0,0)^7-7*(1,2,1,3,4,0,0,0,0,0,0,0,0,0,0,0)^8

}
catch(Exception ex)
{
    Console.Out.WriteLine(ex.ToString());
}

while (true)
{


    int p = 0;

    while (p == 0)
    {
        Console.Out.WriteLine("input p");
        Int32.TryParse(Console.In.ReadLine(), out p);
    }

    Console.Out.WriteLine("p=" + p);

_RESTART_A:

     Console.Out.WriteLine("input A: a_0,a_1,..." + (p - 1) + " coordinates \n press R for random");
```

```
System.String a = Console.In.ReadLine();

System.String[] data=null;
List<BigInteger> list1 = new List<BigInteger>();
List<BigInteger> list2 = new List<BigInteger>();
int l1 = 0;
int l2 = 0;

try
{
    if (a.StartsWith('R'))
    {

        for(int i=0;i<p-1;i++)
        {
            list1.Add(r.Next(-10000, 10000));
        }



    }
    else
    {
        data = a.Split(',');

        //l=p-2
        l1 = data.Length;

        if (l1 != p - 1)
        {
            Console.Out.WriteLine("wrong input");
            goto _RESTART_B;
        }

            for (int i = 0; i < l1; i++)
        {
            string s = data[i].Replace(" ", "");
            BigInteger x = Int32.Parse(s);
            list1.Add(x);

            input += "" + x;
            if (i < l1 - 1)
                input += ",";

        }
    }
}
catch (Exception ex)
{
    Console.Out.WriteLine("wrong input");
    goto _RESTART_A;
```

```
        }

        input += "\n";

    _RESTART_B:

        Console.Out.WriteLine("input B: b_0,b_1,..." + (p - 1) + " coordinates \n press Xn
for (1,...,1,0,...,0) n-times 1");

        a = Console.In.ReadLine();

        try
        {
            if (a.StartsWith('X'))
            {
                int n = Int32.Parse(a.Substring(1));

                for (BigInteger i = 0; i < Math.Min(n,p-1); i++)
                {
                    list2.Add(1);
                }
                for (BigInteger i = Math.Min(n, p - 1); i<p-1; i++)
                {
                    list2.Add(0);
                }
            }
            else
            {

                data = a.Split(',');

                l2 = data.Length;

                if (l2 != l1)
                {
                    Console.Out.WriteLine("wrong input");
                    goto _RESTART_B;
                }

                    for (int i = 0; i < l2; i++)
                {
                    string s = data[i].Replace(" ", "");
                    BigInteger x = Int32.Parse(s);
                    input += "" + x;
                    if (i < l2 - 1)
                        input += ",";

                    list2.Add(x);
                }
            }
        }
```

```
 catch (Exception ex)
 {
     Console.Out.WriteLine("wrong input");
     goto _RESTART_B;
 }
 input += "\n";

 BigInteger [] A = list1.ToArray();
 BigInteger [] B = list2.ToArray();



//Console.WriteLine("input:\n\n" + input);
int e=0;

 while (e == 0)
 {
     Console.Out.WriteLine("input exponent e");
    Int32.TryParse(Console.In.ReadLine(), out e);
 }

 Console.Out.WriteLine("e=" + e);

 BigInteger [] C = Exponent(B, e, p);
 BigInteger [] D= Multiply(C, A, p);

 System.String result = "";

 result+="A=";
 for (int i = 0; i < p - 1; i++)
 {
     result += "" + A[i];

     if (i < p - 2)
         result += ",";
 }
 result += "\n";
 result += "B=";
 for (int i = 0; i < p - 1; i++)
 {
     result += "" + B[i];

     if (i < p - 2)
         result += ",";
 }
 result += "\n";
 result += "RESULT=";
 for (int i = 0; i < p - 1; i++)
 {
     result += "" + D[i];
```

```
            if (i < p - 2)
                result += ",";
        }

        Console.Out.WriteLine(result);

        //check for redundancy
        BigInteger [] E=D.Distinct().ToArray();

        Console.Out.WriteLine("distinct values="+E.Length+"/"+(p-1));

        // Array.ForEach<BigInteger >(D, x => Console.WriteLine("" + x));

    }

  }
}
```