Write-Up Qualification
HOLOGY 2025

jay
Xxymbol
fele

minion cabang depok

DAFTAR ISI

CRYPTOGRAPHY

[200 pts] The Architect’s Hasty Encryption

Description
An emergency ping from the System crackles through your interface. The Architect’s module spins up, whispering of keys tempered in unstable instant magic and relics humming with old power. The gate won’t open itself. Plug in, look around, and figure out what the System wants from you.

Solusi:

Karena diberikan e yang cukup besar, saya langsung curiga ini pasti merupakan wiener attack. Kemudian nyuruh gpt scripting, setelah itu berhasil recover ketiga plaintextnya. Setelah cukup lama stuck, akhirnya nanya probset kemudian dapat ide utk nyari gcd a dan b. Kemudian concat dengan x. Berikut solvernya :

```
from sage.all import *
import hashlib

nM =
Integer(144391215245328813496207595557892054275155586523856267225815345657584232798089835711453017277486236037622965850466965573741587223417088659009396057475097724047043421582370562986500546461394171566317657490044733980974256375311066078804458168359827510546460074362471318929332294458813355366147714019823027129949)
eM =
Integer(739615754189578406698828301088522293950160760711239432464477038505648532686418112106742512948108150419453071235281370029932820737872465983299987254624640983271541715745419784158865065649774878316032650775885286700634517835732292557455497147928742341509629677504077728845453490705613337268057191271182915242 25)
cM =
Integer(790053079249419131839318550085817812029710697868872840124113328579587510950612373416449603051868932827331624009854770686351244381919921271678591190827487770870699731808741100917645572168302470856086770404824326225227700020419648719304825345649233878090896885874708179206978882378651415517380283780318811493 31)
```

```
# Artefacts
nA =
Integer(775987522625755493253646723939307120766063808775925796150934992638386977036703955485961388804289554099603210849371728455115215108470493624612415179772717559572761495468459958804937172293770140676949400593886994212494820609785846937939279496382203691583811681559936005101199847367936908223924962228869051322911)
eA = Integer(65537)
cA =
Integer(515946821048695776694015843054609223876069720858522679751318547703344973263252310286314672581345241906122650584551640912602162095779283385793003628030456725124546436153778073344524231622986734496367405222611323494925003452922760791242481892102488164873816514756769240429772944268305816562629636218041683339 7)

nB =
Integer(593074854155400669237302422479890932487695880377845572216977648325370657291155406136488601436648795006275923057077144085136760695349129272244154638994582114566294784065081648071988666554798715628170754536298767910732224738537791114461461185226345633089578611054528899564862258601043587798928325340103974180791)
eB = Integer(65537)
cB =
Integer(306365305493921597164845981866827348903727480696553853133913878643436424058494914407581037902572595531955748685386834565264268156590126186024530316705347997482834713187683337596431237076466793832587831878677126339137537301745268072649101265235962587471014419740984902219379898624401743269736271810165331331 03)

nX =
Integer(821253598481604547255445116094819336804245976444684912262476775916852336755825031144162031133409668081937782219378374841509859829734211570638599968917450011488056642234316453951710537115824632525947119501928366582988853570539006356660342449794234838669600034657759088502470275941754108467818624000357792396 1)
eX = Integer(65537)
cX =
Integer(725357581812739035162093260421278063890288515835738308561061867853651006943001922554643770483180049387919725071605322227233531203872387833254635743249560470231727549170006433228588841949727880389358806442932975020214332023331682051219661495276706296094221854485307742478815469326769030743546928680742269874 2)
```

```
def i2b(z):
    z = Integer(z)
    return z.to_bytes((z.nbits()+7)//8, 'big')

def sha256_hex(b: bytes) -> str:
    return hashlib.sha256(b).hexdigest()

# 1) Shared prime from artefacts A & B
p = gcd(nA, nB)
assert 1 < p < nA and nA % p == 0 and nB % p == 0
qA = nA // p
qB = nB // p

# 2) Decrypt ART-A & ART-B (optional: just for confirmation)
phiA = (p-1)*(qA-1)
dA   = inverse_mod(eA, phiA)
mA   = power_mod(cA, dA, nA)
plainA = i2b(mA).decode('utf-8', errors='replace')

phiB = (p-1)*(qB-1)
dB   = inverse_mod(eB, phiB)
mB   = power_mod(cB, dB, nB)
plainB = i2b(mB).decode('utf-8', errors='replace')

print("[ART-A]", plainA)
print("[ART-B]", plainB)

# 3) ART-X is a perfect square nX = pX^2 → phi = pX*(pX-1)
assert is_square(nX), "nX must be perfect square (p^2)"
pX = isqrt(nX)
phiX = pX * (pX - 1)
dX = inverse_mod(eX, phiX)
mX = power_mod(cX, dX, nX)
plainX = i2b(mX).decode('utf-8', errors='strict')
print("[ART-X]", plainX)

# 4) Build flag material: ART-X plaintext + decimal p (no separator)
s_concat = (plainX + str(p)).encode('utf-8')
digest = sha256_hex(s_concat)

print("\n[+] Prime (gcd) bitlength:", p.nbits())
```

```
print("[+] Using concatenation:  ARTX_plain + str(p) (decimal, no
sep)")
print("[+] SHA-256:", digest)
print(f"\n=== SUBMIT THIS ===\nHOLOGY8{{{digest}}}\n")
```

FLAG :
HOLOGY8{7ec2c6c4821ffe2bd9fa2045ef69791a22e595425e2948844e9b1fd3b37d54ca}

[456 pts] p-power-rsa

Description
An RSA variant where the modulus has a repeated prime factor. Can you still recover the message?

Author: R1v3r

Diberikan,

```
from sage.all import *
from Crypto.Util.number import *
import gmpy2
import random
from sympy import nextprime


FLAG = b'HOLOGY8{REDACTED_REDACTED_REDACTED}'
m = bytes_to_long(FLAG)
r = random.randint(5, 30)


p = getPrime(256)
q = getPrime(256)
if p < q:
    p, q = q, p
N = pow(p, r) * q
phi = pow(p, r - 1) * (p - 1) * (q - 1)


e = 65537


c = pow(m, e, N)
print(f'c = {c}')
print(f'e = {e}')
print(f'N = {N}')
```

```
d1 = getPrime(2000)
d2 = nextprime(d1 + getPrime(1000))
e1 = gmpy2.invert(d1, phi)
e2 = gmpy2.invert(d2, phi)
print(f'e1 = {e1}')
print(f'e2 = {e2}')
```

```
c =
11148578741621573208341988835605800103397939936210936050826187089114
84072440569153208420570045057827156028181234473008840950208480418896
95814526433868629365023169179222249266389625852952987468768744093949
28518305914109263583529836974227705330538612949449893288884757430080
83880892683252853155447687582439627763053385226221852536414060400397
66863839006857421878923985989934964980608678162026614939501709616234
02256051306872774122745141884319923513574909526934129781982520409344
91520401194504287605833225994118264938664198308269684384412766681923
58564834603596666357837414657021199051732261588602224173516664537564
14839408955042965148380999786745067799292206419334487477074750340090
40938017808736269480924938483778916870232621005585404027295880314360
30432454794249812343438390584037127772193904170421087845937142186548
75974530485345842610889547272654404100971318581885200905937761466010
01113040922140022529357970105883386976827646232264113729591494752134
91898026585839227437083313605739136374826611550285890516268274465906
51602736225060976763088927714433438532990708303684257823041138278098
69034930961352233448908795589570445785916699320291444553085845395422
87253481580691466811698093274707201630462570730775293023489199764838
10352375566246137475119895846742635800668869428611373349947620894937
86972396117163064777826915719498372710037996238034841685083674404660
97349584557388186877058472216663997659060248313992041943213504830112
92871077770270415445375268583670974485113451091685851930681091720325
01486157989738547685799120192688845063502134676407713464851764195822
69169891572750725266499997539100897292414501918122708319038201277181
78087134916043563030557962263828373741474240294496361421437239651613
04999822527648186438619503913331971241426026581892974695018317308377
39445119194832277196410874479162266849124548988494138572551706403876
05610788711387174199311750727941171876289823861267567691315593947944
94188680420063298157018187160344334249364926565204433449440138818534
87228563694901400446844531668
e = 65537
```

N =
45374999360986242376740631549994357671561927071935540297423859805295
53949281157365520521493599977862462203566976820858759529218143036086
98370621001593498202664137105614797706092963766648597009961505999304
80486862881991305383674320335278979004362972200265756125120047946168
23878412099894321149754916241773844518913913791868779434356749413927
93230667987075796005304723586746499234907181428238498088293365909082
01631281787915892649163343882886427956981388044992890319744231405679
80829483163453330201243334222710102665582407044626577954373312845684
70273071623784463980804356407048493506788785174496491522489832266993
85749357114179109582456092820278785391845490692529410810830763245464
31812922842381390141396556614688558576411125274758109643984898177268
86715552651548640655208966527802039076290335379020936965043298763839
40159917151440553306438288924650907511067102750891160442877581241774
03691278060151124778162240900133870550291471107609587274609296125964
88683715157524348089512845563356023108705591279503195318426654042354
99516873082722427577433858935496084389231152779521025662507616153014
80143083735102054758031266202731220263554484311016232518079148726570
94636473462128842533375598676295429191094715741380629940237085984351
82845540267141135904098507938822378845568711349570185162389495695764
55509255199128563782586379754441399685527831592764818262945580264305
14802530631451990707908860581231085522384501312007775067802744550720
12130971156806367205765189457157795566712527784194089612138959747780
09995100998978024585195694883991499678909222606659665158786253841426
99235491857754702310410293104365494144520258022000143265495269730031
16317052964039607035814432505055426282735050697422531659417643183918
28345884572254968633128186074911835384215030850952593817795980833772
26530394482769313660477032407154580895878594632722208659675114254258
70472798001225357792182807761489625539080569789001601132798402922131
15387898067931606626553838499139436472821929593599203139362225802842
59387292748269260001904819421
e1 =
39165793652087577914877426625039278725640736188258425465833695599956
34003034056304607208169618172246187377772942924817928667361463495520
43520851645842130597971547038173283631333584316297965498373376861095
53082999368553839561049565344905533509876796662001162885617336052479
91788955600412275719385038142141591395598454013559838006708085851768
76221210353125537444035036065337535172729006915261241937199553763509
19305455728337463804098470218344001010105494234598392272868848980973
41749309390181713454788411385573672176165277377016330277090481605190
29152593665114170813759658432086356637433620887643733535199031310414

774416867729652384099267600031352379211898316329552882992285392553077527756827101153728363057673373227069245765439725259700104492923535011956521059690189455889183451733852768162679237572669069711674417535640195150268613405492138817491324569277302613361765379500965882363974155431727439589100873409740046911824512781264499781490426245133694125237856572372320276726919978657920936435451576648357500045008986846304153298148480178606891143308643867687679179582600314502641722670246145884011183046072245800750643498168832668146747562640149522944351635221567326443563204062253274832487797124510410876930344635405809024564216083826083606356349920292962209883332800936657765601014586626521147867354148932767989773867054883139306513657206425612731393762848284337977952741882659790830673601398929277057124801924729734092576210233733348524332130803895377108038459257464444108228034148791591771078270957276307142848729206654090569158864178327128965680534718726000268582897902099331752602828329471333188980982758605078545558572617649043450812146592911695959737057539204043200951055437750050392284647477611306914795301712624484309660059522390682543744798410321583111434249958206819770249403779250080423111546721491964159141897836449404286604243936534599014320416032579619418739621845891364003447681178546955999694669399489613255215803856197169937382196973478712792226779871071390136699165091671245266 5
e2 =
391480438310625429700531646158375558956506797101675973418606782333011334646342690789224987096829046860981688968724248619810659493788011346075406331109254895794954604059311803095476184098658711476514271012640430957705253154502948997851491293854297194287496988866498497276886406861821843453141954575964039834749531822831337130185953913373230219695605326376578117028418146781175845432888789627828317850201340638281641866763886764105619790084294728116079020769412002026441350399332180101103670349129744461547719171682711593163835339556804954874495741026848950434966723117849592956416315695901260200650285038247514989373648105175019183344134033846456492491420859603206529031959678368108422616482651192173701656311229490616200075542644714659465573895201229989360138716158652073422542820973786851429541964027946920027948104910877506713855994606116374606525167951425480919451945584455139519911160474700546497875260672336092706569154899095801313465594693841762286492695428917833693360082474466996222871292730722222946290331536403719254010129799842315180984576276859440760856275999356164675709206534753046105671584659337162168237383329104690132391289474285892121493573333706762707079879638517612766161523755581149348087190079207202864710547794996521690018469547788542207870355235680280422024256004684048

421460788165600487113923745754048461068434449755153303076136653667869925596304162973359620352595601031005250955396262548151927859866097970832280797709625301158364314606588220127790116430081995663341647634971071595183484675026199416331961306054378319525288775203633847462948820166267532966102786816849571851084755394048971360722722991499233241222588363003038362465256441612026422078939529054315956083762044326947570467716187320364655071058442648207756640966556148959746013175585365125000210507145407697085781184379840872128156894086697960739781022168256529541675871913015651256275622524733493659109405638687233774546864903638260531315149916767007797660636877896004135827330681187330654206445399574150663933545 5

Kemudian saya menemukan WU dari sebuah blog:
https://www.cnblogs.com/chen-xing-zzu/p/18620187

kemudian saya meminta chatgpt untuk menjelaskan kepada saya bagaimana solvernya bekerja
https://chatgpt.com/share/68d9e0d1-3888-8003-b8ce-275ffb7fcf3c
dan ini adalah kunci untuk modifikasi solver pada WU dari blog tadi

$$p_6 = gcd(e_1 e_2 x + e_2 - e_1, N) = p^6$$

oleh karena itu p^(r-1) = gcd(e1 e2 x + e2-e1, N)
Kemudian, saya terpikirkan untuk for loop aja untuk bruteforce r.
Kemudian saya coba lakukan modifikasi terhadap kode:

```
from hashlib import md5


from Crypto.Util.number import *
from gmpy2 import *
from sage.all import *
c = 
11148578741621573208341988835605800103397939936210936050826187089114840724405691532084205700450578271560281812344730088409502084804188969581452643386862936502316917922224926638962585295298746876874409394928518305914109263583529836974227705330538612949449893288884757430080838808926832528531554476875824396277630533852262218525364140604003976686383900685742187892398598993496498060867816202661493950170961623 4
```

022560513068727741227451418843199235135749095269341297819825204093449152040119450428760583322599411826493866419830826968438441276668192358564834603596666357837414657021199051732261588602224173516664537564148394089550429651483809997867450677992922064193344874770747503400904093801780873626948092493848377891687023262100558540402729588031436030432454794249812343438390584037127772193904170421087845937142186548759745304853458426108895472726544041009713185818852009059377614660100111304092214002252935797010588338697682764623226411372959149475213491898026585839227437083313605739136374826611550285890516268274465906516027362250609767630889277144334385329907083036842578230411382780986903493096135223344890879558957044578591669932029144455308584539542287253481580691466811698093274707201630462570730775293023489199764838103523755662461374751198958467426358006688694286113733499476208949378697239611716306477782691571949837271003799623803484168508367440466097349584557388186877058472216663997659060248313992041943213504830112928710777702704154453752685836709744851134510916858519306810917203250148615798973854768579912019268884506350213467640771346485176419582269169891572750725266499997539100897292414501918122708319038201277181780871349160435630305579622638283737414742402944963614214372396516130499982252764818643861950391333197124142602658189297469501831730837739445119194832277196410874479162266849124548988494138572551706403876056107887113871741993117507279411718762898238612675676913155939479449418868042006329815701818716034433424936492656520443344944013881853487228563694901400446844531668
e = 65537
N =
453749993609862423767406315499943576715619270719355402974238598052955394928115736552052149359997786246220356697682085875952921814303608698370621001593498202664137105614797706092963766648597009961505999304804868628819913053836743203352789790043629722002657561251200479461682387841209989432114975491624177384451891391379186877943435674941392793230667987075796005304723586746499234907181428238498088293365909082016312817879158926491633438828864279569813880449928903197442314056798082948316345333020124333422271010266558240704462657795437331284568470273071623784463980804356407048493506788785174496491522489832266993857493571141791095824560928202787853918454906925294108108307632454643181292284238139014139655661468855857641112527475810964398489817726886715552651548640655208966527802039076290335379020936965043298763839401599171514405533064382889246509075110671027508911604428775812417740369127806015112477816224090013387055029147110760958727460929612596488683715157524348089512845563356023108705591279503195318426654042354

995168730827224275774338589354960843892311527795210256625076161530148014308373510205475803126620273122026355448431101623251807914872657094636473462128842533375598676295429191094715741380629940237085984351828455402671411359040985079388223788455687113495701851623894956957645550925519912856378258637975444139968552783159276481826294558026430514802530631451990707908860581231085522384501312007775067802744550720121309711568063672057651894571577955667125277841940896121389597477800999510099897802458519569488399149967890922260665966515878625384142699235491857754702310410293104365494144520258022000143265495269730031163170529640396070358144325050554262827350506974225316594176431839182834588457225496863312818607491183538421503085095259381779598083377226530394482769313660477032407154580895878594632722208659675114254258704727980012253577921828077614896255390805697890016011327984029221311538789806793160662655383849913943647282192959359920313936222580284259387292748269260001904819421e1 =391657936520875779148774266250392787256407361882584254658336955999563400303405630460720816961817224618737777294292481792866736146349552043520851645842130597971547038173283631333584316297965498373376861095308299936855383956104956534490553350987679666200116288561733605247991788955600412275719385038142141591395598454013559838006708085851768762212103531255374440350360653375351727290069152612419371995537635091930545572833746380409847021834400101010549423459839227286884898097341749309390181713454788411385573672176165277377016330277090481605190291525936651141708137596584320863566374336208876437335351990313104147744168677296523840992676000313523792118983163295528829922853925530775277568271011537283630576733732270692457654397252597001044929235350119565210596901894558891834517338527681626792375726690697116744175356401951502686134054921388174913245692773026133617653795009658823639741554317274395891008734097400469118245127812644997814904262451336941252378565723723202767269199786579209364354515766483575000450089868463041532981484801786068911433086438676876791795826003145026417226702461458840111830460722458007506434981688326681467475626401495229443516352215673264435632040622532748324877971245104108769303446354058090245642160838260836063563499202929622098833328009366577656801014586626521147867354148932767989773867054883139306513657206425612731393762848284337977952741882659790830673601398929277057124801924729734092576210233733348524332130803895377108038459257464444108228034148791591771078270957276307142848729206654090569158864178327128965680534718726000268582897902099331752602828329471333188980982758605078545558572617649043450812146592911695959737057539204043200951055437750050392284647

```
477611306914795301712624484309660059522390682543744798410321583111434249958206819770249403779250080423111546721491964159141897836449404286604243936534599014320416032579619418739621845891364003447681178546955599969466939948961325521580385619716993738219697347871279222677987107139013669916509167124526652
e2 =
391480438310625429700531646158375558956506797101675973418606782333011334646342690789224987096829046860981688968724248619810659493788011346075406331109254895794954604059311803095476184098658711476514271012640430957705253154502948997851491293854297194287496988864984972768840686182184345314195457596403983474953182283133713018595391337323021969560532637657811702841814678117584543288878962782831785020134063828164186676388676410561979008429472811607902076941200202644135039933218010110367034912974446154771917168271159316383533955680495487449574102684895043496672311784959295641631569590126020065028503824751498937364810517501918334413403384645649249142085960320652903195967836810842261648265119217370165631122949061620007554264471465946557389520122998936013871615865207342254282097378685142954196402794692002794810491087750671385599460611637460652516795142548091945194558445513951991116047470054649787526067233609270656915489909580131346559469384176228649269542891783369336008247446699622287129273072222294629033153640371925401012979984231518098457627685944076085627599935616467570920653475304610567158465933716216823738332910469013239128947428589212149357333370676270707987963851761276616152375558114934808719007920720286471054779499652169001846954778854220787035523568028042202425600468404842146078816560048711392374575404846106843444975515330307613665366786992559630416297335962035259560103100525095539626254815192785986609797083228079770962530115836431460658822012779011643008199566334164763497107159518348467502619941633196130605437831952528877520363384746294882016626753296610278681684957185108475539404897136072272299149923324122258836300303836246525644161202642207893952905431595608376204432694757046771618732036465507105844264820775664096655614895974601317558536512500021050714540769708578118437984087212815689408669796073978102216825652954167587191301565125627562252473349365910940563868723377454686490363826053131514991676700779766063687789600413582733068118733065420644539957415066393354555
```

```
R = PolynomialRing(Zmod(N),'x')
x = R.gen()
f = e1 * e2 * x + e2 - e1
```

```
f = f.monic()
roots = f.small_roots(X = 2 ** 1000,beta = 0.7)
#print(roots)
if roots:
    p_r_1 = GCD(int(e1 * e2 * roots[0] + e2 - e1), N)
#print(p_r_1)


for i in range(5, 31):
    r = i+1
    p = iroot(p_r_1,r-1)[0]
    #print(p)


    q = N // p ** r
    #print(q)
    if q == 0: continue
    flag = long_to_bytes(pow(c,inverse(65537,pow(p, r -
1)*(p-1)*(q-1)),pow(p, r) * q))
    if(b'HOLOGY' in flag): print(f"flag: {flag}")
```

Dan ketika dijalankan

Flag : HOLOGY8{Mu17i_P0w3R_RSA_w17h_W34k_Struc7ur3}

WEB

[100] pyjail?

Description
I hope I patched all the unintended ways...

Solusi:

Sepertinya probset masih belum patch unintended yang ada pada chall, atau memang LFI adalah vuln yang sebenarnya dan bukan pyjail. Karena saat mengakses http://ctf.hology.id:8282/flag.txt, flag langsung diberikan begitu saja.

HOLOGY8{h0l1_m0l1_r3g3x_g07_pwn3d??_51ff8a6c}

[100] Hidden Factory

Description

Welcome to the "Hidden Factory"! A secretive manufacturing facility has protected their critical blueprints using an intricate multi-layer encryption system. The factory's security engineers have implemented a sophisticated data transformation pipeline that processes sensitive information through multiple encoding chambers. Can you unravel their encryption methodology and extract the original blueprint? The solution requires dissecting their security architecture, reversing each protective layer, and recovering the authentic manufacturing documentation.

author: Lorem

`nc ctf.hology.id 1000`

Solusi:

saat connect nc, diberikan

```
❯ nc ctf.hology.id 1000 ⏎
============================================================ 🏭
HIDDEN FACTORY
============================================================ 📋
CHALLENGE DESCRIPTION: Welcome to the "Hidden Factory Blueprint
Vault"! A mysterious industrial facility has secured their most
valuable blueprint using a sophisticated encryption system. The
factory's engineers have developed a multi-layered security
protocol that transforms sensitive data through various
computational processes before locking it away. Can you
reverse-engineer their protection mechanism and recover the
original blueprint? The secret lies in understanding the factory's
methodology, tracing back through their security layers, and
carefully reconstructing the original information.
============================================================ 🎯
YOUR TASK: Analyze this source code and decrypt the encrypted
```

```
blueprint below.
============================================================ 📊
ENCRYPTED BLUEPRINT:
Sd9CG5TC4Xrz6ISW830mryN0qzBXwV/lyRt+cA2ifxAxXJPRAbLYR65qQWzv1HrKAJ
b1CBEwt+F8tvNnYfsBdAoQifQnjNi/GLHB
============================================================
```

Dan diberikan source code yang berupa code Cython panjang yang merupakan module untuk `chall.pyx`. Namun step yang dilakukan untuk decrypt blueprint sebenarnya cukup simpel, yaitu recover custom base64 ke normal base64. Karena pada code diberikan hardcoded value `secret_multiplier = 7` dan `magic_offset = 23`, konversi bisa langsung dilakukan dengan melakukan XOR per-index keystream `((i*7)+23) mod 256`. Setelah direcover, plaintext dapat dicek dengan:

```
import base64

plain =
"HOLOGY8{f4ct0ry_r3v3rs3_3ng1n33r1ng_m4st3r_0f_th3_h1dd3n_bluepr1n
t_cr4ck3r}"
alphabet =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789{}_
"
std =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
cust =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/"

s1 = "".join(alphabet[(alphabet.index(c)+23)%len(alphabet)] if c
in alphabet else c for c in plain)

s2 = bytes([ord(ch) ^ ((i*7 + 23) % 256) for i, ch in
enumerate(s1)])

b64 = base64.b64encode(s2).decode()
enc = "".join({s:c for s,c in zip(std, cust)}.get(ch, ch) for ch
in b64)

print(enc)

//python3 check.py
```

```
Sd9CG5TC4Xrz6ISW830mryN0qzBXwV/lyRt+cA2ifxAxXJPRAbLYR65qQWzv1HrKAJ
b1CBEwt+F8tvNnYfsBdAoQifQnjNi/GLHB
```

```
flag =
HOLOGY8{f4ct0ry_r3v3rs3_3ng1n33r1ng_m4st3r_0f_th3_h1dd3n_bluepr1nt_c
r4ck3r}
```

[100] ObligatoryFlagCheckerThatIsPacked

Description
Title should suffice

Solusi:

Diberikan file executeable packed_rev, karena judul dan nama file adalah packed, yang pertama dilakukan adalah unpack file:

```
❯ upx -d ./packed_rev -o ./unpacked_rev
_⌟

                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2025
UPX 5.0.2       Markus Oberhumer, Laszlo Molnar & John Reiser
Jul 20th 2025

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
[WARNING] bad b_info at 0x4772c

[WARNING] ... recovery at 0x47728

    767128 <-    321328   41.89%   linux/amd64   unpacked_rev

Unpacked 1 file.


~/Doc/Pr/CTF-Journey/R/Ch/ObligatoryFlagCheckerThatIsPacked-hology
25    main !1 ?94
❯ ls
_⌟
packed_rev  unpacked_rev


~/Doc/Pr/CTF-Journey/R/Ch/ObligatoryFlagCheckerThatIsPacked-hology
25    main !1 ?94
```

```
❯ file unpacked_rev
⎵⌟
unpacked_rev: ELF 64-bit LSB executable, x86-64, version 1
(GNU/Linux), statically linked,
BuildID[sha1]=35088fa8eff71cd1632788893d1626094304c42b, for
GNU/Linux 3.2.0, not stripped
```

Setelah file di-unpack, dilakukan analisis pada IDA disassembly dimana terdapat tipikal flag checker. Hasil static analysis adalah berikut:
`main` print `WHATS THE FLAG, SEARGEANT.` dan baca 256 char input dengan `fgets(v10, 256, stdin)`.
`v16 = strlen(v10)`, then `v10[v16] = 0x20 /* space */`. `v15` is set to `30`.
Jika `v16 == 30`, lanjut; otherwise program exit.
Builds 30-byte buffer di `v13[i]` sebagai:

- `v7 = partA[i]`
- `v8 = rol8(partB[i], 3)` (8-bit left rotate sebanyak 3 kali)
- `*((BYTE*)v13 + i) = v8 ^ v7`

lalu dilakukan check sebanyak `j in [0..29]`:

- `v12 = v10[j]` (input byte)
- `v11 = key[j & 7]` (key berulang tiap 8 bytes)
- Bandingkan `(v11 ^ v12)` dengan `*((BYTE*)v13 + j)`; jika tidak cocok → `THATS NOT RIGHT`.

jika flag match, print `YAY YOU GOT IT RIGHT`.
Data:

- `partA : E4 4E 5B DD 8B 08 FC 63 CD D2 38 64 17 B4 52 66 8C D6 03 7F 74 97 FC 7C 25 78 6B 2F 8C 6F`
- `partB : 39 0C 8C 7D 72 47 34 2C D8 10 0F 2F 6F 77 0D 65 D6 70 E5 8E 03 51 D8 AE 8E 4F 6E AC 34 2F`
- `key = b"easy_key"`.
- `rol8(x,3) = ((x << 3) | (x >> 5)) & 0xFF`.

Maka tinggal dilakukan reverse pada tiap part untuk recover flag:
`flag[i] = key[i & 7] ^ (partA[i] ^ rol8(partB[i], 3))` for `i = 0..29`, menghasikan:

HOLOGY8{n33d3d_4_4_r3v_p4ck3r}