

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації**

**Звіт до
комп'ютерного практикуму №3**

Оформлення звіту:
Дигас Богдан, ФІ-52мн
Юрчук Олексій, ФІ-52мн

6 жовтня 2025 р.
м. Київ

Комп'ютерний практикум № 1

1.1 Вступні відомості

Мета роботи: Ознайомлення з підходами побудови атак на асиметричні криптосистеми на прикладі атак на криптосистему RSA, а саме атаки на основі китайської теореми про лишки, що є успішною при використанні однакового малого значення відкритої експоненти для багатьох користувачів та атаки "зустріч посередині", яка можлива у випадку, якщо шифротекст є невеликим числом, що є добутком двох чисел.

Постановка задачі:

1. Створіть репозиторій у системі контролю версій Git/GitHub;
2. Реалізувати атаку з малою експонентою на основі китайської теореми про лишки.
3. Реалізувати атаку "зустріч посередині" та порівняти її швидкодію з повним перебором можливих відкритих текстів.
4. Оформити звіт до комп'ютерного практикуму.

Результати виконання роботи. Варіант 15

1.2 Small exponent attack

1.2.1 Принцип алгоритму:

Користувач надсилає однакове повідомлення M кільком k користувачам A_1, A_2, \dots, A_k , $k \in \mathbb{N}$, причому відкритий ключ користувача $A_i, i = \overline{1, k}$ – це пара чисел (n_i, e) , де e – деяке мале число. Тобто юзери A_1, A_2, \dots, A_k мають однакову експоненту e , яка використовується для шифрування. Тоді відправник зашифрує своє повідомлення M , використовуючи відповідні пари (n_i, e) для кожного з отримувачі $A_i, i = \overline{1, k}$.

Зловмисник підключається до каналу передачі даних і перехоплює шифротексти C_1, C_2, \dots, C_k , де $C_i = M^e \bmod n_i, i = \overline{1, k}$. Вважаємо, що значення n_i для $i = \overline{1, k}$ попарно взаємно прості. (Інакше можна було б обчислити \gcd і факторизувати, як мінімум, два модуля $n_i, n_j, i \neq j$, обчислюючи $\gcd(n_i, n_j)$). А також, що $M < \min \{n_i | i = \overline{1, k}\}$. Тобто маємо такий випадок:

$$\begin{cases} C_1 = M^e \bmod n_1; \\ C_2 = M^e \bmod n_2; \\ \dots \\ C_k = M^e \bmod n_k. \end{cases}$$

Якщо $k \geq e$, то зловмисник, перехопивши значення C_1, \dots, C_k , може дешифрувати повідомлення M наступним чином:

1. Обчислює значення $C = M^e \mod (n_1 \cdot n_2 \cdot \dots \cdot n_k)$, використовуючи китайську теорему про лишки;
2. Оскільки $M < \min \{n_i | i = \overline{1, k}\}$, то $M^e < n_1 \cdot n_2 \cdot \dots \cdot n_k$. Тоді $C = M^e$.
3. Обчислюємо корень m -того степеня і знаходимо оригінальне повідомлення: $M = \sqrt[m]{C}$

1.2.2 Results of SE

Згідно варіанту було взято спершу тестовий dummy-варіант, а потім, власне, real завдання. Також для порівняння часу виконання, було реалізовано додаткову функцію яка обраховує корені n -того степеня методом Ньютона (метод дотичних).

Отримані наступні результати:

```
-----
Getting values from: '../var_data/test_SE_RSA_256_var15.txt'
Small Exponent attack started
N1 = 78173750441896293770589248403106329516337365403796571933408209730624448306381
C1 = 15609656025847217715374515923704532133385823587903747495107568196099489477819
N2 = 83486112986036005959284637373037544575249353832575223997220194899962725393289
C2 = 10551343233045069430747535369849363227483605229269791213276377761239511954354
N3 = 111166333229840961378827817264163380896109621083905439048552336846491866489619
C3 = 51085873664558514528356006501757009171408402270556826115363936105786995225051
SE message: 3533694129556768659070856691244179748522658885726707029670113186869509344
SE message in hex : 1fffffffffffff001d309c33b6c7ff52cb35beb9c75b4b656586dd7ce0
'Small exponent' execution time: 2055.5 mu s
```

Рис. 1.1: На тестових даних

```
-----
Getting values from: '../var_data/SE_RSA_1024_5_hard_var15.txt'
Small Exponent attack started
N1 = 13041674159766450573472652217629247864516182891999787795138794048454698974615426355469875783752976594300612666940791174216241868330346281750617105198904988951399
C1 = 89769243445475691563782970751164097260547046507901589653632095124636860288746607312197562893304117752905010233677885850062994611809313485557884542985462630654621
N2 = 117539161655176827686141125083899386301787486762925259277489248073893835072745846830735924294803765194816045073898248616481849981414756356735825607629085346760
C2 = 7365512631932471589853405937655943713065408672898288414740691844022694594447207375518432757385623404002875294375258618614079431543541221475096200800155250
N3 = 1659247983419898029129543559287676944203621533987488875719965299964304676575332439955370836373344953814439342910912410774072460548957473152968553799944252706716
C3 = 46474199433923478114775220545318614495271045371755266999157907561789963187253589900118591022755738109775681483654881059743813182126179307346803098841560491752785
N4 = 15692730404439546959883773152765036932196748536890263377904188563947301514551113983962954483925362587904131872193444811558999325563778803612412198932819060522688
C4 = 12719536407291918235824008674911278003718954618043855369219730222118479215329067191380037178384568490250504704386621433869473262396821806513262464273984687276295
N5 = 13898824511393712566135010452918049323241276511344025116632407371106392502554903177315514566468742828440791109309753098315502000953215897075226577898371734370388
C5 = 88956532117284290981967781579244249148826833959801765956924653828647621831552889947806775939291083754252600803581947109543267600240216894673779937967576446226
N6 = 30328029089574081756018110631023008760076590475837295706269706295039143869236601076211424875823167285465220083166177237523719802250442729284018773
SE message: 5486124068793688683107314286044525818244550380220780198621805133840290447640268623617983887494485539736501671089378089679026787815734737539621503499236430
C6 = 14129951166575736858776713813861554256752566995902778279820942070690432200449646284681225642446851768270887178891453428050054740417076496466121830197
SE message in hex : 1fffffffffffff00233064fa9d923b83f6c109e3ba09c7c050e06639aaf1f43334d7d6f98404923ad8c662190fc73b3fa4713cd6af12d9c918be3085645b7fc435785c421276a97
3b32018ff95e8a2cd6a2110f4e83abb7a603d75e7982391c67334167c9436564bd8430e8625073cb6edfb469cbcea4f6c8e827af35
'Small exponent' execution time: 72082.3 mu s
```

Рис. 1.2: На стандартному варіанті, з custom $\sqrt[n]{}$

```

-----
Getting values from: './var_data/SE_RSA_1024_5_hard_var15.txt'
Small Exponent attack started
N1 = 1304167415976645057347265221762924786451618289199978779513879404845469897461542635546987578375297659430061266694079117421624186833034628175061710519890498
89513991570458316565313625114458273829326025276933247498804411883344236711830335281963913676855098808069097399575836065845111761820907547477563875397497643
C1 = 89769243454756915637829707511640972605470465079015896536320951246368602887466073121975628933041177529050102336778858500629946118093134855788454290546263
0654621040408315408176610806838200721476842488920359042603271451620571857159592044183835637052110001482255958664262105804054114401240272760865878767545935
N2 = 1175391616551768276061411250583098306301787486762925259277489248073893035072274584683073592429480376519481604507389824861648184998141475635673582560762908
53467601969249288514960100916319462223890005599223469159064811433872466229966033789886659458843693392791750485986500899299396668015402092003143013258866089
C2 = 7365512631932471589853405937655943713005405540067289982288414740691844022694594447297375518443275738562349409287529437525861861407943154354122147509620080
015525058088652325993359548121652572871143169733549873264966725037395594748982070897363938489463187823426779016509196259199901232738136182540771357633103
N3 = 1659247983419098092129543559287676944203621533987488875719965299964304676575332243995537083637334495381443934291091241077407246054895747315296855379994425
27067164457595356745269367861163426507069788429520548914402116043163580440246163410372545676703514648920663156334679132030155402060366838919456211109141853
C3 = 4647419943392347811477522054531861449527104537175526699915790756178996318725358980011859102275573810977568148365488105974381318212617930734680309884156049
1752785059024400088067974389609248328581430602850089733117501301995636527857528659597477517232537319530448118517180755345524942844555690279290686216615164
N4 = 1569273040443954695988377315276503693219674853689026337790418856394730151455111398396295448392536258790413187219344481155899932556377880361241219893281906
05226883742150072321670186289745700236391417557288822683013626101293727862255040461950692875489992903553370435521149931333661139626273929471223761063137277
C4 = 1271953640729191823582400867491127800371895461804385536921973022211847921532906719138003717838456849025050470438662143386947326239682180651326246427398468
727629561025246885144955988575164142610432935761821889043506026701346680404264075766819348749490623349049069716332106762843406118194942278471203051670536
N5 = 1389882451139371256613501045291804932324127651134402511663240737110639250255490317731551456646874282844079110930975309831550200095321589707522657789837173
437038886676213425044119025757499476534685044800397792951768897102120593292078471967728812259150250240793511520639862293294649232685625227908492192888622191
C5 = 8895653211738429098196778157934434914882603395989126505692465382864762218315620890478067759392010837543252600803503194710954326700924021689467377993796757
64462263823802908057480175601911063102109871600765904750372957706629592913438692366910762114248758216728546322008031661772752719802250442729204018773
SE message: 548612460793680683107314260644525818244550380220708100671005133840290447640938026179038074040485530736501671080370080697907678157473753062150349
92264301412095116657573685877671381386155425675256960959027782790209420706904320044064628460122564246851768270807178091453428059054748417076496466121830197
SE message in hex : 1fffffffffffffffff00232064fa0d023b03f6c100e30a90c7c050e06639aaf1f4334d7d6f08040923ad8c662190fc73b3fa4713cd6af12d09c918be3085645b7fc435785c42
1726a073b2018ff95e8a2cd6a2110f4e83abb7c608d7e5e7982391c67334167c9436564bd8430e8625073cb6edfb469cbcea4ffc6e827af35
'Small exponent' execution time: 22176.4 mu s

```

Рис. 1.3: На стандартному варіанті, з `inbuild` $\sqrt[3]{C}$

Для даного виду атаки, різниця часу у зв'язку з застосуванням іншої функції обчислення $\sqrt[3]{C}$ не є суттєвою, вона складає $\sim 50000 \mu s$, що дорівнює усього $0.05 s$. Як можна бачити, стандартна реалізація є більш вдалою.

1.3 Man(Meet) in the middle attack

1.3.1 Принцип алгоритму:

Зловмисник перехопив деякий шифротекст $C = M^e \bmod n$, причому відомо, що $M < 2^l$, $l \ll \log_2 n$. З доволі великою ймовірністю повідомлення M є складеним числом, тобто його можна представити як добуток чисел $M_1 \cdot M_2$. Нехай при цьому $M_1 \leq 2^{l/2}$ та $M_2 \leq 2^{l/2}$. Тобто можемо розписати:

$$C = (M_1 \cdot M_2)^e \bmod n = M_1^e \cdot M_2^e \bmod n.$$

Зловмисник дешифрує повідомлення M , виконуючи наступні кроки.

1. Криптоаналітик формує множину пар X виду:

$$X = \{(1, 1), (2, 2^e \bmod n), \dots, (2^{l/2}, (2^{l/2})^e \bmod n)\}$$

Кожна пара має вигляд: $(T, T^e \bmod n)$, де $T = \overline{1, 2^{l/2}}$

2. Далі послідовно обчислює такі значення:

$$C_S = C \cdot S^{-e} \bmod n, \quad S = \overline{1, 2^{l/2}},$$

де $S^{-e} \bmod n$ – попередньо обраховані значення з множини X .

(а) Для кожного значення C_S , одразу після його обчислення, зловмисник шукає в множині X таку пару, щоб $S = (T^e \bmod n)$ для деякого значення $T = \overline{1, 2^{l/2}}$.

(б) Якщо таке T не знайдено, повертаємось на крок 2 та обчислюємо наступне значення C_{S+1} . Якщо при цьому $S = 2^{l/2}$, то алгоритм дешифрування зупиняє роботу і виводить "Відкритий текст не визначено".

3. Для знайденого значення $T^e \bmod n$ виконується рівність:

$$T^e = C \cdot S^{-e} \bmod n.$$

Тоді маємо:

$$C = (T \cdot S)^e \mod n,$$

тобто шифротекст C було отримано внаслідок шифрування відкритого тексту $T \cdot S$

1.3.2 Results of MitM

Також згідно варіанту було взято тестовий dummy-варіант для MitM, а потім, власне, real завдання. Отримані наступні результати:

```
-----
Getting values from: '../var_data/test_MitM_RSA_256_var15.txt'
> MitM (optimized) attack started for:
l = 56
N: 100039008232565778030511827934443693881873768818632993921733760872111379080792863325428540036282008104493965528664727
99792929866363242122358904236296789077
C: 378706066813226414875579047252487295283243408295027138466466599554965038889175627344324878386423045824052823567801601
1090597562892504184637396054614419136
Start: bn_t = 0, bn_s = 0 : 71
Generated: bn_t = 0, bn_s = 0 : 190958104
MitM message: 425079
MitM message in hex : 67c77
Optimized 'Meet in the middle' execution time: 2.85559e+08 mu s
```

```
-----
Getting values from: '../var_data/MitM_RSA_2048_20_regular_hard_var15.txt'
> MitM (optimized) attack started for:
l = 56
N: 259184146537004307610829237988106804772837188562507139305915304689282063752456996295814599082339878915899244151969020
997425935706030233676545600833751815080442497423996571705433224827129481638564294857125889017489932948312704767428346365
643592537205591638640693079296429124619892458053799110404126860354066048967963210822899570932576193549612174086962466286
300921682880177151937211376971952280544893345324682201719617813244216767730994505789180971543210985669540088948251480817
675264879954136786967328399292610731395201725823776142098567278594552620919467305935173899522941298667043410839269444955
84215958472944393123
C: 901857421401009352117314080877294551717694888475755975119907071775249401691416230597369434433456348351398490878996395
347296015761162362862591020527428935946975980648693351946008209345580514465731887535972736191091277614736858076391792668
323255630109510105302972943821911643125525291471473772318815590251995849464611105993541058726658005560932397610132750877
939117362375852014541072616724056815525363223244932454305420695496211178176310423755815770477439546857670408295123779369
253700077125458156350870569979322443311339207398963180241922468307804231128500989403385677482561657441628233364539602121
4664439255174831040
Start: bn_t = 0, bn_s = 0 : 82
Generated: bn_t = 0, bn_s = 0 : 398159551
MitM message: 551361
MitM message in hex : 869c1
Optimized 'Meet in the middle' execution time: 8.11774e+08 mu s
```

Також ми зробили тест bruteforce, який повідомляє час на 1 спробу підбору.

```
-----
Getting values from: '../var_data/MitM_RSA_2048_20_regular_hard_var15.txt'
Bruteforce once try (find next M & get C & make a compare) time: 62.6 mu s
```

Помноживши кількість можливих варіантів ключів 2^{2048} на $62.6 \mu s = 62.6 \times 10^{-6} s$ маємо, що час на злам прямим перебором займатиме:

$$T \approx 2.02 \times 10^{612} \text{ seconds} \approx 6.41 \times 10^{604} \text{ year}$$

Ця цифра є фактично нескінченністю (порівняно з часом життя Всесвіту який дорівнює $1.3798 \times 10^{10} \text{ year}$) для будь-яких практичних задач. Навіть за умови надзвичайно оптимізованого паралелізму (використання мільярдів ядерів) brutforce 2048-бітного ключа RSA є абсолютно нездійсненним.

Далі ми спробували (і змогли!) зробити бонусне завдання додавши оптимізацію. Його результат є наступним:

```
-----
Getting values from: '../var_data/bonus_MitM_RSA_2048_56_var15.txt'
> MitM (optimized) attack started for:
l = 56
N: 236712988205543636185673865821355696997789889997966211196357840289554616636179490362135049929073540232779656343895739
00463929860953658266939762761346960369540177376034628798399759328077165584017133370158933506453095492550328035538765846
482873046456403462779596083807424489795343807003985321849870047985556479700428107321119453497955135844956932970402576598
108012890832373622311761898674445795143043242270549912923823829771326932774793490648005088035544251229249223109501652608
326293273282183573073935348785043028577869518291298677402599654566163703201434172829792421473665511969667505818727571941
73418535716416432961
C: 183127077133937877306841119098923081020721608705078106968095985413379356657675703284680628148305160792031719835359959
041194515818806744390222433718876394240138038647960238415407786471780694644770824300368974054305648753017340703816307320
218571655694927399545439698394338252653290461631031496680783943721118032303756139820930344988645544793129539406743534790
684412474526379665633632295463235726498703501326976775624531566766132334364176699101227282425507215060280377405285511924
617398941802638387517726638503541634251733924249251135871012761008196140773364620575923494519716583685022479008075796811
49172446012303944985
Start: bn_t = 0, bn_s = 0 : 108
Generated: bn_t = 0, bn_s = 0 : 357858418
End: bn_t = 0, bn_s = 0 : 2523434251
Start: bn_t = 0, bn_s = 1 : 2523436573
```

```
Start: bn_t = 7, bn_s = 13 : 78261695516
End : bn_t = 7, bn_s = 13 : 79083201966
Start : bn_t = 7, bn_s = 14 : 79205300528
End : bn_t = 7, bn_s = 14 : 79974989491
Start : bn_t = 7, bn_s = 15 : 80108769415
End : bn_t = 7, bn_s = 15 : 80934994295
Start : bn_t = 8, bn_s = 8 : 81211898059
Generated : bn_t = 8, bn_s = 8 : 81356984059
End : bn_t = 8, bn_s = 8 : 81733447713
Start : bn_t = 8, bn_s = 9 : 8173344811
End : bn_t = 8, bn_s = 9 : 82558653035
Start : bn_t = 8, bn_s = 10 : 82694620858
End : bn_t = 8, bn_s = 10 : 83519495327
Start : bn_t = 8, bn_s = 11 : 83646916347
End : bn_t = 8, bn_s = 11 : 84417995063
start : bn_t = 8, bn_s = 12 : 84556185711
End : bn_t = 8, bn_s = 12 : 85379475837
Start : bn_t = 8, bn_s = 13 : 85506823206
End : bn_t = 8, bn_s = 13 : 86282524062
Start : bn_t = 8, bn_s = 14 : 86422391454
MitM message: 34537772074549159
MitM message in hex : 7ab3ebb3d63fa7
Meet in the middle with space compromise execution time : 8.68417e+10 mu s
```

Процес зайняв порядку 24.5 годин.

1.4 Висновки:

Завдяки комп'ютерному практикуму ми розібралися з двома атаками на криптосистему RSA, "побачили руками, очами потрогали" як вони реалізовується на практиці. Було важко розібратися з оптимізацією алгоритму-атаки MitM. За допомогою документації C++ ми змогли реалізувати оптимізацію, яка економить ресурси комп'ютера. Довелося робити trade-off крок в бік оптимізації під простір і розбити простір перебору на блоки, генерувати їх, очищати, робити перебір і генерувати знову.

Також були думки серіалізувати список T , $T^e \bmod N$, а також були думки з вивантаженням його в постійну пам'ять, але нас зупинили думки про I/O overhead та складна програмна реалізація відповідно. Також була ідея з використанням іншого API [arrayfire](#) для пришвидшення роботи алгоритму.

P.S. На жаль, найбільш наївне рішення проблеми не спрацювало :(

<https://downloadmoreram.com/>