

Міністерство освіти, науки, молоді та спорту України

Національний технічний університет України

«Київський політехнічний інститут»

Фізико-технічний інститут

**АСИМЕТРИЧНІ КРИПТОСИСТЕМИ**

**ТА ПРОТОКОЛИ**

**КОМП’ЮТЕРНИЙ ПРАКТИКУМ №3**

**Криптосистема Рабіна**

**Атака на протокол доведення знання без розголошення**

**Виконали:**

Студенти групи ФІ-73

Морозюк Анастасія,

Гетьман Дмитро

**Перевірив:**

Деркач О. Г.

Київ 2020

**Мета**

Ознайомлення із криптосистемою Рабіна та особливостям її реалізації. Ознайомлення з криптографічними протоколами взагалі та протоколами доведення знання без розголошення зокрема. Ознайомлення із перевагами, недоліками та особливостями реалізації різних криптографічних протоколів. Аналіз наведеного протоколу; реалізація атаки на цей протокол.

**Завдання**

Комп’ютерний практикум включає в себе два завдання: реалізацію розширеної криптосистеми Рабіна та реалізацію атаки на описаний протокол доведення із нульовим розголошенням.

Реалізація криптосистеми Рабіна виконується таким саме чином, як і реалізації криптосистеми RSA у комп’ютерному практикумі №2. Основні операції (генерування ключів, шифрування/розшифрування, постановка/перевірка підпису) необхідно оформлювати високорівневими процедурами GenerateKeyPair(), Encrypt(), Decrypt(), Sign(), Verify().

Кожну операцію рекомендується перевіряти шляхом взаємодії із тестовим середовищем, розташованим за адресою

http://asymcryptwebservice.appspot.com/?section=rabin .

Наприклад, для перевірки коректності операції шифрування необхідно а) зашифрувати власною реалізацією повідомлення для серверу та розшифрувати його на сервері, б) зашифрувати на сервері повідомлення для вашої реалізації та розшифрувати його локально.

Для проведення атаки на протокол доведення із нульовим розголошенням також необхідно користуватись тестовим середовищем.

За адресою http://asymcryptwebservice.appspot.com/?section=znp проживає сервер, який генерує ключі RSA довжиною 2048 біт та користується описаним протоколом, щоб довести будь-кому своє знання розкладу модуля на прості множники.

1. Реалізуйте допоміжне програмне забезпечення для проведення сценарію атаки.

2. Згенеруйте на сервері ключі для аналізу. Сервер поверне вам значення модуля *n* (це значення буде існувати доти, доки ви не завершите сесію зв’язку).

3. Користуючись формою введення, надсилайте серверу випадкові *t*, поки атака не завершиться успіхом. Зафіксуйте, з якої спроби вам вдалось зламати ключ.

4. Продемонструйте викладачеві вашу перемогу над бездушною машинерією.

**Хід роботи**

**1. Згенеровані числа, які не пройшли перевірку**

**Generated numbers for p:**

33E78028A3663B52B84926D7E60B11DBD3E847FE0BAA2C77BDAAF1AF5A5C665C

5A7A90B8132CA5F8E76D14FE2B72B28DA0F569697C78F8E715056B4256CE8907

163744354D9BAC81488BE19CF6FF143435BC54654536945DB099AE5BF04997B0

2977FD3E7821D09C24316F25EA6FD3135B53788D7BB29700FA9538204401E709

4AA0117142AB42CC04F71E9BF84815D198A55C8A9866A753AA2EAC245F3B5DFF

724BF67EE7BB43214E5371590B61D9166A90B708FF2E3A09AABB40DB1FAF3AF0

7E4C795C197CD3478A1CD2D1F59EE71386F673210A00E28CDC220E06C558D810

324E60084B89A02C44AE00C2397E0F58B21E7F590161C91C04AF8DAEE2444E54

51C82680FED2A77136C05BACCFB3BED457EC3B8259A679D901C6F101FF20C54F

154FD1178BDFCED870ACE7D957B2DEC99D574F2A697D81663C0C80AE2CB0863

643224EA4BE34A98F97CCC1213BDB85C99D8D42044B67D392F85251602620205

310789D9CCF6440EEB6DCE40EF548C88FCBECF0B03DE840C8CDDA6B81468C777

**Generated numbers for q:**

0C3BBB3570C321D94CBD5AFF55CAA0E36B5446868304A895CE70EC554DF5CC4E

2D7264F5DEDB637A2178476585EB92102812F4636BFF078458107C3E1564F5B1

3713588D248BAD922EC12E030A3D48F0F0CE7415211F09F5FE241BF55B536A38

4202AC96969E7EA22F29F3AB2F974EA3C469D5903EA05D9D621B008A05052148

63C11C783FA877A69DB8BAA4C702E44E3EC3AD9AFDE9F411541F7AD232135750

7B41228E53281A5E0E2697085EFA7B358FF7185E91D9C8CA5428C13698164B26

1EB8D38A000695D31997D5092B94BF9C92975B483B61400EFAC612BAA0D37B1C

78434E2BF0245E0CD0489B6C92C6DCFED0092B09D23C9A20B9E536A248A18689

65367F49980E7E203CC7D3135808F5B78AEDBA0DADCC14FA83A20CFCF2547823

76B5F28E105E1C45C3F793D7977AB7B07333A2666C55D9A9682EB71FF9E770C7

5757873783090D7FE1E5CA1254F884F5F2B46C5202CF99F96A8257C54979B06A

5DB6758808C9CC4879194C91EFD5AC272368049C001419AF2D1D20CDB5208A3E

290629555CCA492FF474748B7A685DA231944E51414277DBF59058E4671C6846

504A6B8B5AECCACBFE96F68891C44A96DA0BC3CADDDCF2A959FE433B24868922

4DD808A80FF9FA1FE6C179A434F92A168EA0C3AAEF185BD355AC2FF3FEE82312

6E2D73247D7CA4A3811354BD4695497FC78127DF7A35631D3D94AFCD2D4724D5

276EE023131B8D02204743B6F7909610EF6767C8DC98597D3309D2D544BFD602

0F5298E7E24B85415E55DA48995F2DEC9EA1CEE9DDEF909289AFA41C0829F68F

4A1424B0F4E9C23E07F6FADAF09CADE558C7092B92157DEA64F3ED0AFEDDF695

0E3B968830E7C4E648061FD7C468A5A5F18F69E84D870571E4F6B1F7428864EA

632D1110BF6F81986195F493B5C584C4FEED054CF84521D39138C09F6140AC69

61CD674AB7C6B9750FE6FD800F17D72A0D3946CB759346A03758015EF474A382

450674EAB699F4FA92A97C89C87A5E70488FAAA805E170DC5FACC2769E778FAF

216861239E46F63787F739F82387632B8AC495F5F7049CAF7190619F8D548243

443C3C10C42E4A8663ED450EB80C73402DEF26A81F3AE59BA3588A625504747D

507CE3CA52F3919EB2BB8C3E31AADF71B94EA33C5D28BF8C01DE99F69AE9974E

49314680369B136EE093943CD590909E00FC76050CC9900F9FCFD3E6BFB99E5E

0849DB60EDE52479F73C8B0EE87AE9578E6F82DDF115F839155DD7EF7C07513

0BFD7B2BB34E4A30E2EEE0A5AB889BA12E6CCD6BED8E849836760E4692F8EA80

6D0ED73FCE24C0391ADB1A13F7672CCFB814ABC57E9F54A7F2C7FD10C27823BD

6604150A32C38CCD801132383D386F655AC3131E17C9392B562CB07BB7AEFCFC

5F2EDB587C2581EB6BA4172B0B9FC1A1BFDB6B04FFC9C870492ACE873963ED8C

4C031561E5EE36EC1DDE2AE94843DC9F697AB90986BBB9BDC564C9769E68CBB6

7A0B90FF58CB7C410C0BE930CDB21BAB187AC2D3ABF93699B85C4470676AC1E1

13695B9272504F2DE69B5C08A084F22F5741ECA11089F66FE6180995D612C27D

7998F1CA3242156A4F3227DFAAF8855ACE727F428B13A3657D6B47DF1D9FC00

1D9D62E8077369E5F8D9524E36934B9DA7FD4F4FC7DF7F909B9FA4062DB6677C

60877152E7E9137DFD754072EBE502964165DAF4B240CCD3A9F4D6074445ACA1

0F41ADFC152C021C89A6C4B9CBD90EF0DFCBC0F441DAF4627BBD476EB4CD7A2D

4A789130C18547A87E531AE6467F3C1C8E90DC371E54F5C4B9A494B88DD5F95A

09CD96446F4F1B2586170491DA403035A2C86550D805B881D97034E23FFFFF7E

7474E7E9172111E43D312FA5DDA579BE80BA6459DECF2C318A3295A3EAE454B3

613819430E0D11AC483893042ACEBBC4150F5DA385C7524E480D3E385FDE0080

12DD3A0DECFD051C8DBE7B2B9C0A82CEA93EC853A17BA4B442BA06CC9661FA57

09C2538D380545A9C6DBA4B4517A87B0FBF4CB498743EED1A9A8BFE6E3DFD5EC

1889C2618572E9D8CA4F422C1D20316FFDDA79C2C0DE444D64F390485B05ACB5

672FC9DC6868B2C39DF04775EBBD022D6764A3E067FBE5AA9C3972EEFD24ECF9

1BF50B8613D46E0DA82D07484206C051176EDD1889F25EB217CD4CEB9C499F37

1BED2711B6409A96CBE4B88BEF5F4ADD32F67A4B7850B9D00423FA338A78FF0A

6585DC829201FEB2D482109015C4711569B5AB6079A6A03CBC7BB76EB68FF98A

421338824CC61A976C9D349F8F091EF19948B1B729653978EFC4F25FBD7307F5

6C785CFC30658FE6637B3C2BB6E968F52E12DB55925E80079C2548B64FF8023B

3905C7B162B0219510D8E924DADA3EAF108CE7B1A5893402276375C043C443AC

6465117751BE667813425027B70DBF8883FD5E190F3F0146A5617FBC1ED71287

0BC3A3A20F4DE93EE29DC828C58E4D5EB6296909E1BF1FDA777498F1CFC138F9

741DD8D3F4DC0914DA20EAB87A33A5817C44229A2F6A59814BB5BB4EE12A7C9B

7E56C1AA7B208CF87C22FE4367FA929FC5800B4066CE2645764BBD5241B3E4BA

5CB35F1359A97CDFC35328C60A92E37CF14454E6C7E03129B549447E76FA9369

1C3E51A91EBDAAECDD29AFF235C30EB6BECC668502EABF7A1721DA9409E9064D

457223627D32291109873097962219E2212E91A15824147B5175CB97DDA2E125

1973B9E108FCE0FED172B3108D27B5C1BB53A8210C21A6429D454B7FBD8EB217

34A63D41B0EF7CC0E9B443E52F94788BB76C0AB93F01E810FC182219E61500D4

32BE7212A23F8B186AB60E628E00AD148A5848623550E342311A5AD11BE48B03

2C8E4B8CA09EAE396832550965C79B87A087AEA6646F84FA5115887098A6F2F1

36C064C28A3F9716A7793B926F2D2A7EADD17E02FBFF7FB24FCB4A8CC17D5BCA

71DA4219B7E218B852DB7C2D3D2423BD8FC173F8024222CAE23ACD67A3679C96

3BEB02510D2DE3B4CF4701A11CCA29EC81BCCAAE2206F46F32D1B68B6E41A5A7

37C1C399FB1D66C861AB792178F27DF7C4CC9DCFDCE8A77314D99739837A81E0

464A5F76C59D061E349FFCB708F4FA2FE90563794E5073FE07BC5A145F75660F

1E9CF927840C3CDED0D0EA8EB732983A221C0D7421309B765EC38AD871252F0F

312969EC45AB0F9E6E49DA898AAA5077B9277B5B36EB1F962C09C3B5FF09AC94

7E67425539E5079A23FA67E80F9F62F655DE96DE2DD358DA7BEDADA005640AA5

15DE53B8C8A3E724FBB0FB7B467F5081B3A2E65CC0FDB968BE7B93C659C08CC5

0C39EAF28EF1C0E3A3683D085D58055B2D9168FC1A86B2D313D777A721AF0D13

7284FA3826039D361D48FE9653FF4AE46F07B461EA29DA5E2AC2B6374A5F7F92

623AB1588D989D102B6728EF336F2F239E422E2F953A6C5A6B7C49B9B667EF19

3EE6796F600F2C4EE33B1C5F78360FC2FD6D506046408DAF6866B23DF5D8F095

5FA39ED921099522FCFD7C49F34CD8F7BD2BA2911502AA05E485FAF50261CA97

5BE850578E7E06FD7991B7EC9B02A1C38BA33AC68B7861D7DF44DE4AB4986767

173CD4ABDF14AA408BE431E7EF067EF9C238C53020B62E0603D8C85AE5ED7811

67E025C4D592A39AFB0801F4AA6529BE325F2C257CEE10C27516B624859B428C

2BC026742C62DBE260BFCF22680A645FE48FC2A00CD34D3A234A6AE0D07198C6

232B0CEDE0AC2405EA47D372AAFB7B5DEA6D86B1643F517DE67D1EF76033EC9C

671E1940EE9A47C2D5CD49CACED44FCD5CC4E0178950AB0C48F9D4D3691185DE

5DD449F4A54C5DB29D0761823E7DCE72773D2BA7613B5D356F372ED35285A590

74ACECCA304220CC25D76703F7847992F8418FE9DE9754ABEAA9A23687B317EE

72631DF1A3F2983109C571ADD9D9383DBDF6A607DD21D82A8C48EFB52D8E5988

61139A20EC795ABFB0CC2B7EF0D2ECEEF5824DB1CC2341079F4948F09500915C

1C0D1A1CF6F323E2975C3A73BB999F82246414EB19186A409F7E3B7ECE7A43C8

6FB38FBBC8FDCD3992E1B7B9C87CB072AD2BAFF6DDE883549232E21FBA116358

26EF591D91B465155AAFE4B54A0873D8B41253D2C2FF40C74E493F6A2E83593C

5ADA3840C742398C5F54718457B36E9EF1463E37F8AAC87645E844D50253B540

211FE4092FEBA409AEFBF4C8550A2E3ACA63A5390BB489D56A14EDDC12670972

101213F8F2D9E05E9F371161A781D49C26A157F199992E8C13C9BC2F12E46707

362D344C703D6217C837A42BF0F82221F71ED9C797C7F56336654F95E196750C

3E2B5147F314DEDA0D0E1154CD276A09CA01A3E1C92355863E4CFB6A913DBB92

6FDB86FF1B7BBB993E57B12DF710D0575FEE4940004EF40E3CB046696AC71072

2B5EBC62CAF4FC2E81E01775472FCCFE75E0C87160EC2826D91BF64FC688D3FA

68030A6E463B6F2205BD244A17D6E68F9CAC91F3EE873592E8D5D4BBA6A97B04

6ACE57A7A5A594B38779D9B8F7B8B48361F1379C1BEEBB3960E5DE11817120EA

79C76586363ED1C27B822FCF66CD9E3A19DAB4E4C95C3F4E779D84D3746716

1880B2B1B799F4BE0349C388CF8FA7B95C5BADD853F24FB1EBFE3923BCBD5CAF

21C8AAF2E906B30BC92B2155873B05F74B0ECCE68A842E6A617D7B8A2B3557FC

41BC39EAB473883443039A53FA22AB09B9F42B70FF1215D7FC791F80C36558D

6BBB5D5DBB6ED37986CD351629A1E6737AC82F8E59B5C709AC9F3B87C0E1E0CB

5BE680D714CBD1463B8911A20809140099AED1BCD441EF49CEA9F33DFF1E9C9D

5E75819E86A051F665E878D0F7D0D102A1CCA846EAEC90F986FBF2F8B7712316

2D3881411C611A74CF4C4072BA217B22CA6FF1929197DE1BE9E6433C0DDA15EA

1AC2B2396F98FC65A2884D16823BFDAD00620E1705C040C24E1143F2FFD1F5AB

19D7C1A0A8A9FDF08875DDCBCE0C483C8A4ECF13C0080C729D6C5265CD9876D1

5904EA7ECB604AB2C263E274754CE39721378A986A5ECFAFB218F1192797C074

317709092120F9A036832FCCCE174CAFFAC057624555A49EECBF97E918635CE5

69395F8CDBAD0EF90B9B6E6ABA8D65DCD22FDA94F2897C6FA6B1CF4C4314D9DE

082443D92BEBB808E2C462007259261246A0D089AD856AD9AD2A260C46A56159

17952B76D45CB7C4959E7E23F21B1CF31D4988A17B5DFD720BE81A2FAC6FB877

15130A6BC6CDC270D498AFA235F03EC475D809FC72F41791FE787C40245446F3

3B86FBCA6D94AADD42890436F7E9B7469FA71258820DBCD12D81696A523A95A7

3729FCE0B03091F7344CCF553F8933028EE9E2C053A01C50860520BD901161A4

732F4702D7DC4E4B854B890B2A9DD374CFB56E0062960131C0CEDE7F32BB626D

683EB97F2FAF11EC94B6A19FBCDC110D8EA72C47153745C2ED10611992F303E4

4313A06F66C766F57B53FA2B019BEA761298B2D9F86CA9DD0C1898035175F6D9

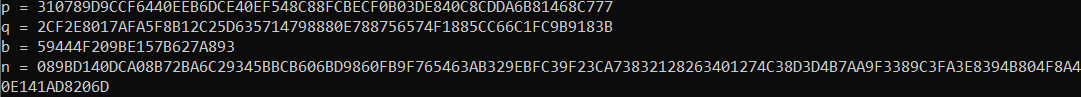
741AF2B24FBAC3B5371377FC0233D940C0216B5A6D891130FF838CFC206DE890

58CDE6F748289366AEF136185453F95C057FBE2E37BA4DA5F67BC60A4612EFC

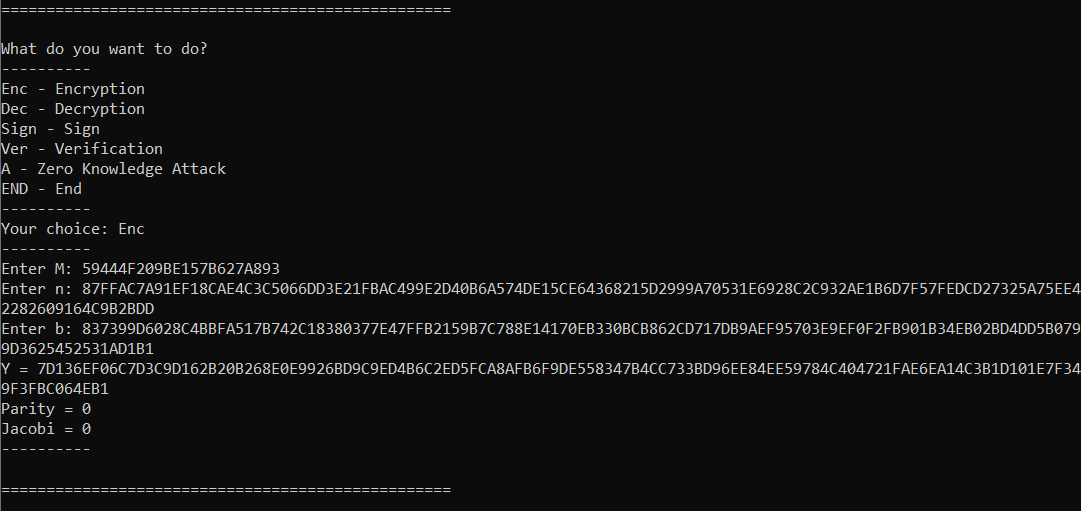
6108CAE456A8457B12DB8E24BA654DA6A20830550E6740CC1426B115C38A6B59

666B18A9FA69831B595D20143A0F20EDA47902E527AFA386C8551238B4955D1D

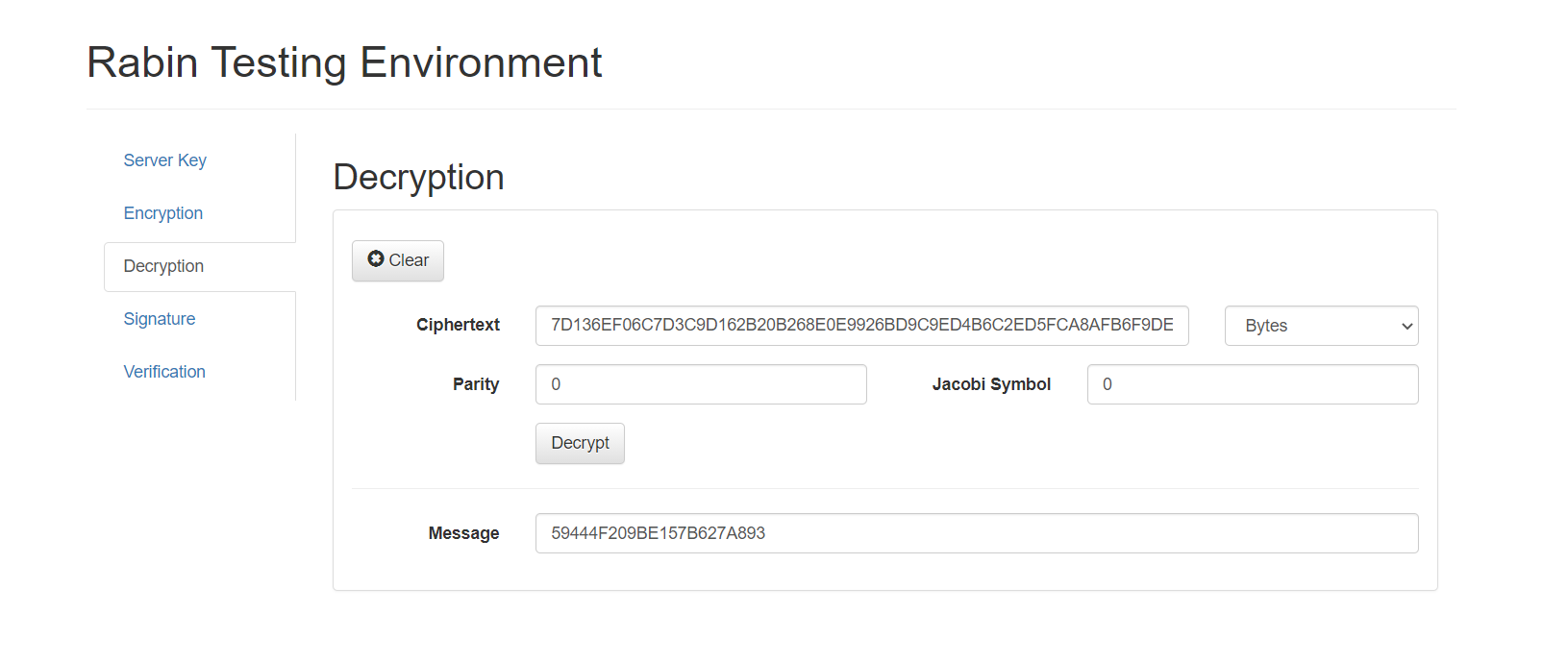
**2. Rabin**



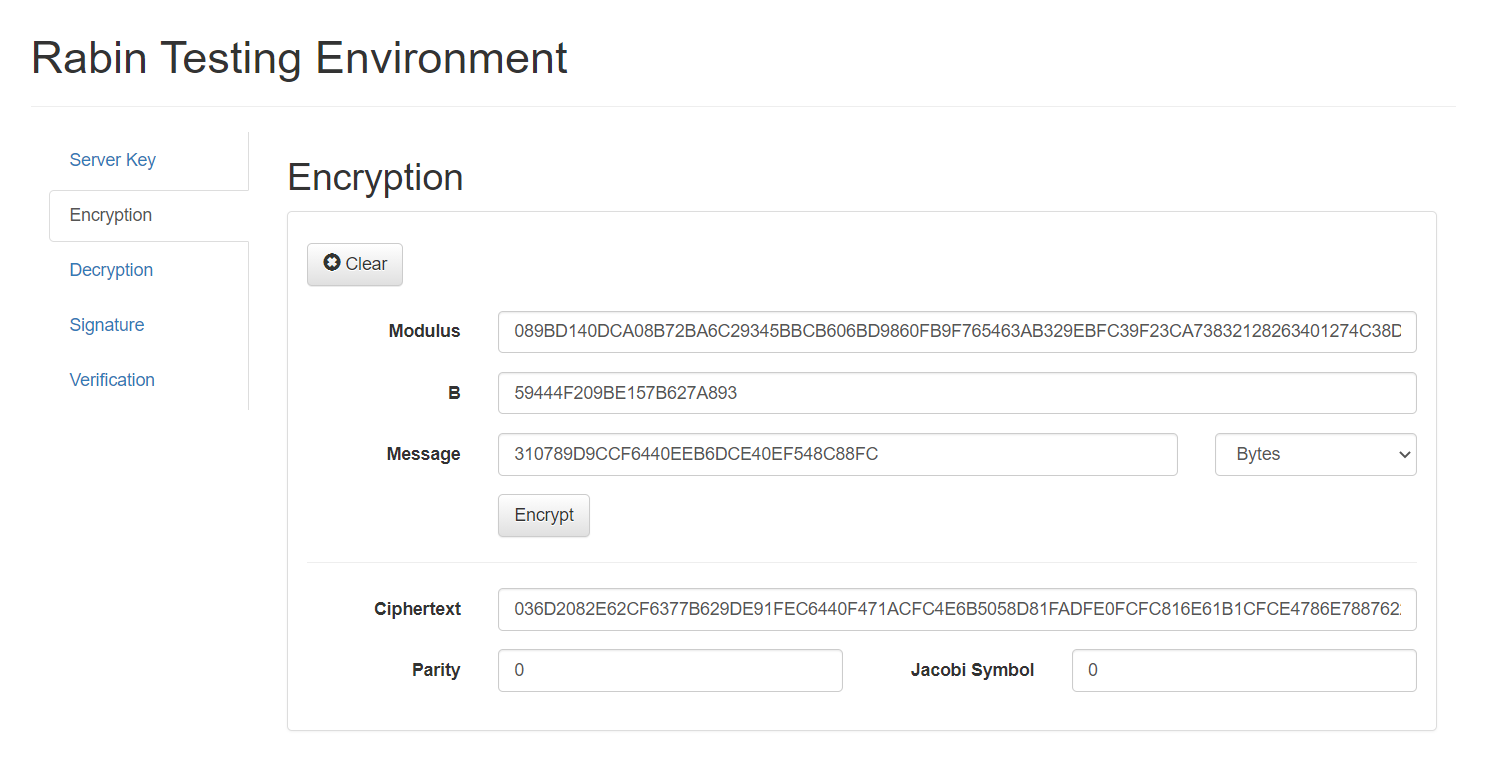
Мал. 1: Параметри абонента А.



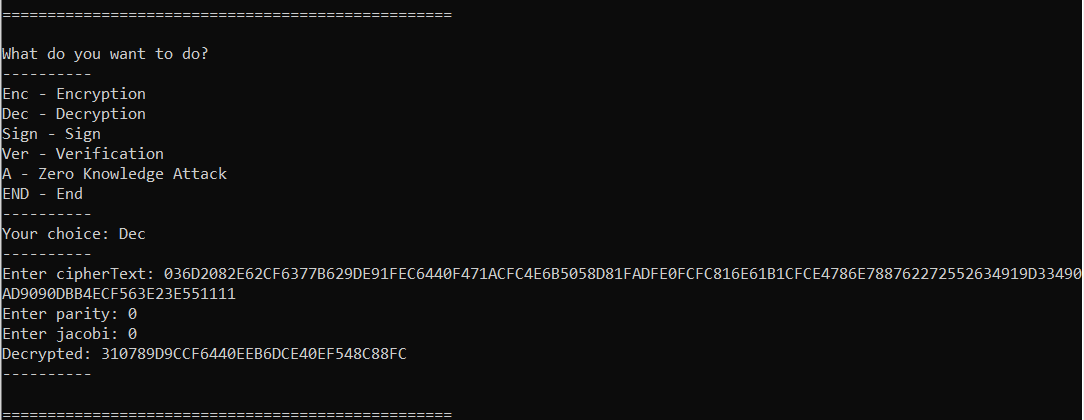
Мал. 2: Зашифрування повідомлення абонентом А.



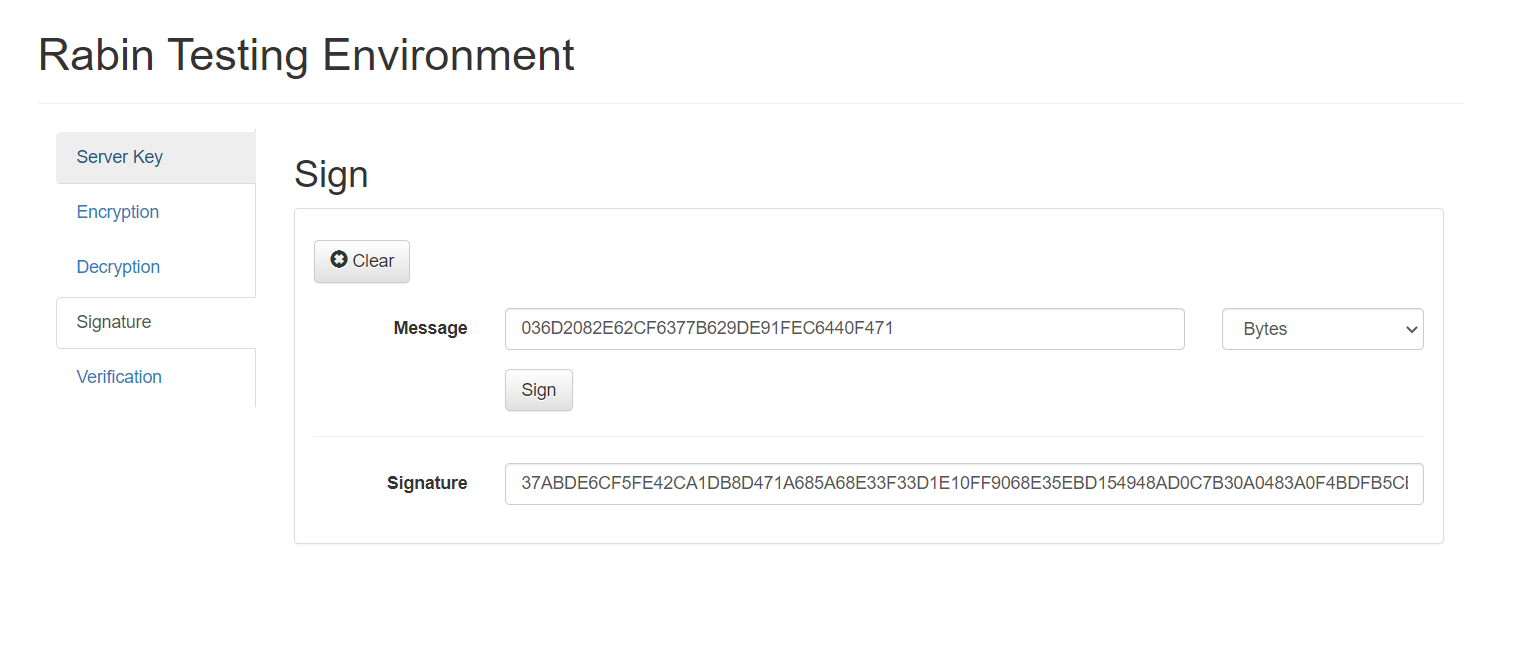
Мал. 3: Розшифрування повідомлення сервером.



Мал. 4: Зашифрування повідомлення сервером.



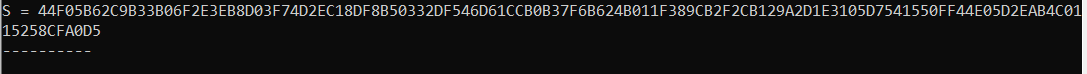
Мал. 5: Розшифрування повідомлення абонентом А.



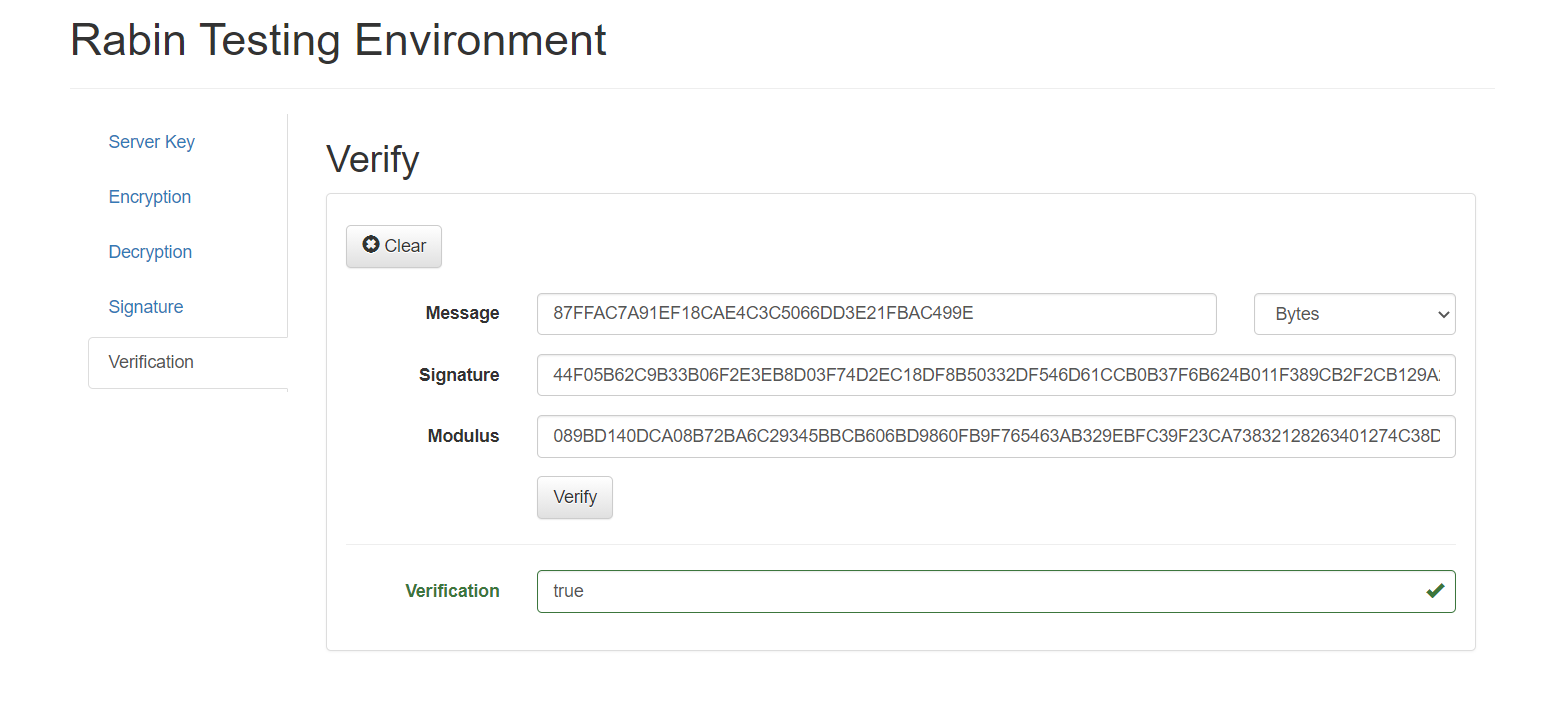
Мал. 6: Підпис повідомлення сервером.



Мал. 7: Перевірка підписа абонентом А.



Мал. 8: Підпис повідомлення абонентом А.



Мал. 9: Перевірка підписа сервером.



Мал. 10: Zero Knowledge Attack.

**3. Опис труднощів**

В ході виконання роботи в нас виникла проблема з функцією розшифрування, в випадку коли с2 = 0, в нас неправильно відбувалося шифрування. Проблема полягала в тому, що ми в фунцкії розшифрування, після знаходження 4 коренів, кол перевіряли умови, який саме з коренів нам підходить повівнювали с2 з символом Якобі від ..., а порібно було порівнювати з символом Айверсона від ..., оскільки символ Якобі може приймати 3 значення (1, 0, -1), а с2 (та відповідно символ Айверсона) - 2 значення (1, 0). Також ми додали перевірки на ненегативність в функцію пошуку коренів, щоб переконатись, що корені точно, знаходяться правильно. В данній роботі також використовували власну функцію зчитування hex-рядка, щоб уникнути проблем,які виникали в комп'ютерному практикумі 2.

**4. Висновки**

Ознайомились із криптосистемою Рабіна та особливостям її реалізації. Ознайомились з криптографічними протоколами взагалі та протоколами доведення знання без розголошення зокрема. Ознайомились із перевагами, недоліками та особливостями реалізації різних криптографічних протоколів.

**Код програми:**

using System;

using System.Numerics;

using System.Linq;

namespace Asym\_Crypto\_Lab\_3

{

class Program

{

static BigInteger TWO = new BigInteger(2);

static BigInteger THREE = new BigInteger(3);

static BigInteger FOUR = new BigInteger(4);

static BigInteger r = GeneratePrime(8);

static BigInteger Inverse(BigInteger num, BigInteger mod)

{

BigInteger q, r, t, u1 = BigInteger.One, u2 = BigInteger.Zero, v1 = BigInteger.Zero, v2 = BigInteger.One,

a = num, b = mod;

while (b != BigInteger.Zero)

{

q = a / b;

r = a % b;

a = b; b = r;

t = u2;

u2 = u1 - q \* u2;

u1 = t;

t = v2;

v2 = v1 - q \* v2;

v1 = t;

}

if (u1 < BigInteger.Zero)

{

u1 += mod;

}

return u1;

}

static Tuple<BigInteger, BigInteger> GCD\_Ext(BigInteger num1, BigInteger num2)

{

BigInteger q, r, t, u1 = BigInteger.One, u2 = BigInteger.Zero, v1 = BigInteger.Zero, v2 = BigInteger.One,

a = num1, b = num2;

while (b != BigInteger.Zero)

{

q = a / b;

r = a % b;

a = b; b = r;

t = u2;

u2 = u1 - q \* u2;

u1 = t;

t = v2;

v2 = v1 - q \* v2;

v1 = t;

}

return Tuple.Create(u1, v1);

}

static BigInteger JacobiSymbol(BigInteger a, BigInteger n)

{

BigInteger r, temp;

BigInteger t = BigInteger.One;

while (a != 0)

{

while (a % 2 == 0)

{

a /= 2;

r = n % 8;

if (r == 3 || r == 5)

{

t = -t;

}

}

temp = a;

a = n;

n = temp;

if (a % 4 == n % 4 && n % 4 == 3)

{

t = -t;

}

a %= n;

}

if (n == 1)

{

return t;

}

else

{

return 0;

}

}

static BigInteger IversonSymbol(BigInteger a, BigInteger b)

{

BigInteger result = JacobiSymbol(a, b);

if (result == BigInteger.One)

{

return new BigInteger(1);

}

else

{

return new BigInteger(0);

}

}

static Tuple<BigInteger, BigInteger, BigInteger, BigInteger> BlumSqrt(BigInteger y, BigInteger p, BigInteger q)

{

BigInteger O = BigInteger.Zero;

BigInteger s1 = BigInteger.ModPow(y, (p + BigInteger.One) / FOUR, p);

BigInteger s2 = BigInteger.ModPow(y, (q + BigInteger.One) / FOUR, q);

var n = p \* q;

var uv = GCD\_Ext(p, q);

BigInteger u = uv.Item1;

BigInteger v = uv.Item2;

BigInteger x1 = (u \* p \* s2 + v \* q \* s1) % n;

BigInteger x2 = (u \* p \* s2 - v \* q \* s1) % n;

BigInteger x3 = (-u \* p \* s2 + v \* q \* s1) % n;

BigInteger x4 = (-u \* p \* s2 - v \* q \* s1) % n;

return Tuple.Create(x1 > O ? x1 : x1 + n, x2 > O ? x2 : x2 + n, x3 > O ? x3 : x3 + n, x4 > O ? x4 : x4 + n);

}

static string HexToBin(string hexString)

{

string binString = "";

for (int i = 0; i < hexString.Length; i++)

{

if (hexString[i] == '0') { binString += "0000"; }

if (hexString[i] == '1') { binString += "0001"; }

if (hexString[i] == '2') { binString += "0010"; }

if (hexString[i] == '3') { binString += "0011"; }

if (hexString[i] == '4') { binString += "0100"; }

if (hexString[i] == '5') { binString += "0101"; }

if (hexString[i] == '6') { binString += "0110"; }

if (hexString[i] == '7') { binString += "0111"; }

if (hexString[i] == '8') { binString += "1000"; }

if (hexString[i] == '9') { binString += "1001"; }

if (hexString[i] == 'A') { binString += "1010"; }

if (hexString[i] == 'B') { binString += "1011"; }

if (hexString[i] == 'C') { binString += "1100"; }

if (hexString[i] == 'D') { binString += "1101"; }

if (hexString[i] == 'E') { binString += "1110"; }

if (hexString[i] == 'F') { binString += "1111"; }

}

return binString;

}

static BigInteger ParseHex(string hexStr)

{

var res = BigInteger.Zero;

foreach (char c in HexToBin(hexStr))

{

res <<= 1;

res += c == '1' ? 1 : 0;

}

return res;

}

/\* Prime numbers generating \*/

static BigInteger GenerateBigInteger(BigInteger max)

{

Random rnd = new Random();

byte[] maxBytes = max.ToByteArray(true, false);

byte[] seedBytes = new byte[maxBytes.Length];

rnd.NextBytes(seedBytes);

seedBytes[seedBytes.Length - 1] &= (byte)0x7F;

var seed = new BigInteger(seedBytes);

while (seed > max || seed < TWO)

{

rnd.NextBytes(seedBytes);

seedBytes[seedBytes.Length - 1] &= (byte)0x7F;

seed = new BigInteger(seedBytes);

}

return seed;

}

static bool MillerRabinTest(BigInteger num, int k = 30)

{

if (num == TWO || num == THREE)

{

return true;

}

if (num < TWO || num % TWO == BigInteger.Zero)

{

return false;

}

BigInteger d = num - BigInteger.One;

int s = 0;

while (d % TWO == BigInteger.Zero)

{

d /= TWO;

s++;

}

for (int i = 0; i < k; i++)

{

var a = GenerateBigInteger(num - TWO);

var x = BigInteger.ModPow(a, d, num);

if (x == BigInteger.One || x == num - BigInteger.One)

{

continue;

}

for (int j = 0; j < s; j++)

{

x = BigInteger.ModPow(x, TWO, num);

if (x == BigInteger.One)

{

return false;

}

if (x == num - BigInteger.One)

{

break;

}

}

if (x != num - BigInteger.One)

{

return false;

}

}

return true;

}

static byte[] GenerateRandomByteSeed(int size)

{

Random rnd = new Random();

byte[] seed = new byte[size];

byte[] zeros = new byte[size];

Array.Fill(zeros, (byte)0);

rnd.NextBytes(seed);

if (seed.SequenceEqual(zeros))

{

seed[seed.Length - 1] = (byte)1;

}

return seed;

}

static BigInteger GeneratePrime(int byteLength)

{

var bytes = GenerateRandomByteSeed(byteLength);

var num = BigInteger.Abs(new BigInteger(bytes));

Console.WriteLine("Generated numbers: ");

while (!MillerRabinTest(num))

{

bytes = GenerateRandomByteSeed(byteLength);

num = BigInteger.Abs(new BigInteger(bytes));

Console.WriteLine(num.ToString("X"));

}

return num;

}

static BigInteger GenerateBlumNumber(int byteLength)

{

var num = GeneratePrime(byteLength);

while ((num - THREE) % FOUR != 0)

{

num = GeneratePrime(byteLength);

}

return num;

}

/\* Rabin cryptosystem \*/

static Tuple<BigInteger, BigInteger, BigInteger, BigInteger> GenerateKey(int PbyteLength, int QbyteLength)

{

var p = GenerateBlumNumber(PbyteLength);

var q = GenerateBlumNumber(QbyteLength);

var n = p \* q;

var rand = new Random();

var randomBytesLength = rand.Next(1, n.ToByteArray().Length);

var b = GenerateBlumNumber(randomBytesLength);

return Tuple.Create(p, q, b, n);

}

static BigInteger FormatMessage(BigInteger m, BigInteger n)

{

int l = 64;

if (l - 10 < m.ToByteArray().Length)

{

Console.WriteLine("message is too big for this n");

return BigInteger.Zero;

}

else

{

var x = r + (m << 64) + (new BigInteger(255) << (8 \* (l - 2)));

return x;

}

}

static BigInteger DeformatMessage(BigInteger x, BigInteger n)

{

var l = n.ToByteArray().Length;

var xStr = x.ToString("X");

var xWithoutR = xStr.Substring(0, xStr.Length - 16);

return ParseHex((xStr.Length == 128 ? xWithoutR.Substring(4) : xWithoutR.Substring(3)).TrimStart('0'));

}

static Tuple<BigInteger, BigInteger, BigInteger> Encrypt(BigInteger x, BigInteger b, BigInteger n)

{

var halfOfB = (b \* Inverse(2, n)) % n;

var y = (x \* (x + b)) % n;

var c1 = ((x + halfOfB) % n) % 2;

var c2 = IversonSymbol(x + halfOfB, n);

return Tuple.Create(y, c1, c2);

}

static BigInteger Decrypt(Tuple<BigInteger, BigInteger, BigInteger> cipherText, Tuple<BigInteger, BigInteger, BigInteger, BigInteger> key)

{

var halfOfB = (key.Item3 \* Inverse(TWO, key.Item4)) % key.Item4;

var temp = (cipherText.Item1 + BigInteger.ModPow(halfOfB, TWO, key.Item4)) % key.Item4;

var sqrts = BlumSqrt(temp, key.Item1, key.Item2);

BigInteger x = BigInteger.Zero;

if (sqrts.Item1 % TWO == cipherText.Item2 && IversonSymbol(sqrts.Item1, key.Item4) == cipherText.Item3) { x = sqrts.Item1; }

if (sqrts.Item2 % TWO == cipherText.Item2 && IversonSymbol(sqrts.Item2, key.Item4) == cipherText.Item3) { x = sqrts.Item2; }

if (sqrts.Item3 % TWO == cipherText.Item2 && IversonSymbol(sqrts.Item3, key.Item4) == cipherText.Item3) { x = sqrts.Item3; }

if (sqrts.Item4 % TWO == cipherText.Item2 && IversonSymbol(sqrts.Item4, key.Item4) == cipherText.Item3) { x = sqrts.Item4; }

x -= halfOfB;

return DeformatMessage(x, key.Item4);

}

static BigInteger Sign(BigInteger m, Tuple<BigInteger, BigInteger, BigInteger, BigInteger> key)

{

BigInteger x = FormatMessage(m, key.Item4);

while (JacobiSymbol(x, key.Item1) != BigInteger.One || JacobiSymbol(x, key.Item2) != BigInteger.One)

{

r = GeneratePrime(8);

x = FormatMessage(m, key.Item4);

}

var sqrts = BlumSqrt(x, key.Item1, key.Item2);

return sqrts.Item1;

}

static bool Verify(BigInteger s, BigInteger m, BigInteger n)

=> BigInteger.Compare(DeformatMessage(BigInteger.ModPow(s, TWO, n), n), m) == 0;

/\* Zero Knoledge Protocol\*/

static BigInteger ZKPsendY(BigInteger n, BigInteger x)

=> BigInteger.ModPow(x, FOUR, n);

static BigInteger ZPKrecieveYsendZ(BigInteger p, BigInteger q, BigInteger y)

{

BigInteger z = 0;

var sqrts = BlumSqrt(y, p, q);

if (JacobiSymbol(sqrts.Item1, p) == 1 && JacobiSymbol(sqrts.Item1, q) == 1) { z = sqrts.Item1; }

if (JacobiSymbol(sqrts.Item2, p) == 1 && JacobiSymbol(sqrts.Item2, q) == 1) { z = sqrts.Item2; }

if (JacobiSymbol(sqrts.Item3, p) == 1 && JacobiSymbol(sqrts.Item3, q) == 1) { z = sqrts.Item3; }

if (JacobiSymbol(sqrts.Item4, p) == 1 && JacobiSymbol(sqrts.Item4, q) == 1) { z = sqrts.Item4; }

return z;

}

static bool ZPKrecieveZ(BigInteger n, BigInteger z, BigInteger x)

=> BigInteger.Compare(BigInteger.ModPow(x, TWO, n), z) == 0;

/\* Zero Knoledge Protocol Attack\*/

static BigInteger ZKP\_Attack\_sendY(BigInteger n, BigInteger t)

=> BigInteger.ModPow(t, TWO, n);

static BigInteger ZKP\_Attack\_getDivider(BigInteger z, BigInteger n, BigInteger t)

{

if (t == z || t == -z)

{

Console.WriteLine("Could not find p or q");

return BigInteger.Zero;

}

else

{

var pq = BigInteger.GreatestCommonDivisor(t + z, n);

if (pq == BigInteger.One)

{

Console.WriteLine("Could not find p or q");

}

return pq;

}

}

static void Main(string[] args)

{

var rndm = new Random();

var keys = GenerateKey(32, 32);

Console.WriteLine("p = " + keys.Item1.ToString("X"));

Console.WriteLine("q = " + keys.Item2.ToString("X"));

Console.WriteLine("b = " + keys.Item3.ToString("X"));

Console.WriteLine("n = " + keys.Item4.ToString("X"));

while (true)

{

Console.WriteLine();

Console.WriteLine("==================================================");

Console.WriteLine();

Console.WriteLine("What do you want to do?");

Console.WriteLine("----------");

Console.WriteLine("Enc - Encryption");

Console.WriteLine("Dec - Decryption");

Console.WriteLine("Sign - Sign");

Console.WriteLine("Ver - Verification");

Console.WriteLine("A - Zero Knowledge Attack");

Console.WriteLine("END - End");

Console.WriteLine("----------");

Console.Write("Your choice: ");

string action = Console.ReadLine();

if (action == "End")

{

break;

}

if (action == "Enc")

{

Console.WriteLine("----------");

Console.Write("Enter M: ");

string M\_hex = Console.ReadLine();

var M = ParseHex(M\_hex);

Console.Write("Enter n: ");

string n\_hex = Console.ReadLine();

var n = ParseHex(n\_hex);

Console.Write("Enter b: ");

string b\_hex = Console.ReadLine();

var b = ParseHex(b\_hex);

var formated = FormatMessage(M, n);

var C = Encrypt(formated, b, n);

Console.WriteLine("Y = " + C.Item1.ToString("X"));

Console.WriteLine("Parity = " + C.Item2.ToString("X"));

Console.WriteLine("Jacobi = " + C.Item3.ToString("X"));

Console.WriteLine("----------");

}

if (action == "Dec")

{

Console.WriteLine("----------");

Console.Write("Enter cipherText: ");

string c\_hex = Console.ReadLine();

var c = ParseHex(c\_hex);

Console.Write("Enter parity: ");

string parity\_hex = Console.ReadLine();

var parity = ParseHex(parity\_hex);

Console.Write("Enter jacobi: ");

string jacobi\_hex = Console.ReadLine();

var jacobi = ParseHex(jacobi\_hex);

var cipher = Tuple.Create(c, parity, jacobi);

var dec = Decrypt(cipher, keys);

Console.WriteLine("Decrypted: " + dec.ToString("X"));

Console.WriteLine("----------");

}

if (action == "Sign")

{

Console.WriteLine("----------");

Console.Write("Enter M: ");

string M\_hex = Console.ReadLine();

var M = ParseHex(M\_hex);

var s = Sign(M, keys);

Console.WriteLine("S = " + s.ToString("X"));

Console.WriteLine("----------");

}

if (action == "Ver")

{

Console.WriteLine("----------");

Console.Write("Enter M: ");

string M\_hex = Console.ReadLine();

var M = ParseHex(M\_hex);

Console.Write("Enter S: ");

string S\_hex = Console.ReadLine();

var S = ParseHex(S\_hex);

Console.Write("Enter n: ");

string n\_hex = Console.ReadLine();

var n = ParseHex(n\_hex);

var ver = Verify(S, M, n);

Console.WriteLine("Verification result :" + ver);

Console.WriteLine("----------");

}

if (action == "A")

{

while (true)

{

int tByteLength = rndm.Next(1, 64);

BigInteger t = GeneratePrime(tByteLength);

Console.Write("Enter n:");

string n\_hex = Console.ReadLine();

var n = ParseHex(n\_hex);

var fauxY = ZKP\_Attack\_sendY(n, t);

Console.WriteLine("y = " + fauxY.ToString("X"));

Console.Write("Enter z:");

string z\_hex = Console.ReadLine();

var z = ParseHex(z\_hex);

var foundP = ZKP\_Attack\_getDivider(z, n, t);

if (foundP != BigInteger.One)

{

Console.WriteLine("P = " + foundP.ToString("X"));

var foundQ = n / foundP;

Console.WriteLine("Q = " + foundQ.ToString("X"));

Console.Write("computed n = ");

Console.Write((foundP \* foundQ).ToString("X"));

Console.WriteLine(foundQ \* foundP == n);

break;

}

}

}

}

Console.ReadKey();

}

}

}