Втор проект по криптографија

Временски напад на RSA

1.Вовед

Во областа на криптографијата, RSA (Rivest-Shamir-Adleman) е еден од најшироко користените криптографски системи за шифрирање и дешифрирање на податоци. RSA се базира на комплексни математички алгоритми и користи факторизација на големи прости броеви за генерирање на криптографски клучеви.

Иако RSA е позната по својата безбедност и ефикасност, во последните години се појавиле нови напади и методи за обратен процес на факторизација. Еден од таквите напади е временскиот напад на RSA.

Временскиот напад на RSA е криптографски напад, кој се фокусира на извлекување на информации со проследување на времето на извршување на операции. Овој вид напад искористува различни аспекти на времето за да ги разоткрие податоците кои се шифрирани со RSA.

Целта на временскиот напад на RSA е да се пресметаат приватните криптографски клучеви, кои се неопходни за дешифрирање на шифрирани податоци. Со успешен напад на овој вид, напаѓачот може да добие приватен клуч и да ги дешифрираат шифрираните информации.

Иако временскиот напад на RSA е комплексен и бара детално познавање на алгоритмот и неговите слабости, се развиваат нови методи и техники за извршување на овој вид напад. Затоа, разбирањето на овој вид напад и негова превенција се критично важни за осигурување на безбедноста на криптираните податоци.

Во овој проект, ќе го истражиме и проучиме временскиот напад на RSA и неговите импликации врз безбедноста на податоците. Ќе се фокусираме на разбирање на нападот, неговите предности и слабости, како и мерки за превенција и заштита против овој вид напад.

2.Техничка обработка

2.1 RSA

RSA (Rivest-Shamir-Adleman) е криптографски алгоритам за јавен клуч кој е искористен за криптирање и потпишување на податоци. Алгоритамот е наречен според трите математичари што го развиле во 1977 година: Рон Ривест, Ади Шамир и Леонард Адлеман. RSA е основан на

математички својства на големи прости броеви и тежи на задачата за факторизација на големи броеви, која се верува дека е тешка за многу големи броеви.

Цел на RSA алгоритамот е да овозможи безбедна комуникација помеѓу две страни преку јавен клуч, кој ќе биде познат на сите учесници, додека само приватниот клуч е познат само на поединците што треба да можат да ја декриптираат или потпишуваат пораката.

Еве како функционира RSA алгоритамот:

Генерирање на прости броеви:

- Избери два различни случајни прости броеви, обележани како p и q. Овие броеви треба да бидат многу големи.
- Пресметај го производот на p и q, обележан како n.

Избор на јавен и приватен клуч:

- Избери еден случаен број e, кој е релативно прост со (p-1)*(q-1), што значи дека нема заеднички делители со него.
- Пресметај го приватниот клуч d, кој е инверзен број на e, модуларно според (p-1)(q-1), односно e * d ≡ 1 (mod (p-1)(q-1)).

Јавниот и приватниот клуч се составени од:

Јавниот клуч: (n, e)

Приватниот клуч: (n, d)

Криптирање на податоците:

Ако "m" е пораката што сакаме да ја криптираме (клартекстот), тогаш ја криптираме користејќи го јавниот клуч (n, e) и следната формула: c = m^e mod n. Тука "c" е криптираниот текст.

Декриптирање на податоците:

Ако "c" е криптираниот текст, декриптираме користејќи го приватниот клуч (n, d) и следната формула: m = c^d mod n. Тука "m" е оригиналната порака.

RSA алгоритамот е особено сигурен за криптирање и потпишување на податоци кога простите броеви p и q се многу големи. Преку факторизација на производот n, што е тешка задача, е тешко да се добие приватниот клуч и да се прочитаат или модифицираат податоците.

2.2 Временски напад

Кошер бил првиот што разговарал за временските напади (timing attacks). Временските напади се форма на "напади со странични канали" каде напаѓачот добива информации од имплементацијата на криптосистемот, а не од било која врска наследена од математичките својства на системот. Ненамерни канали на информации се појавуваат поради начинот на

извршување на операциите или користениот медиум. Нападите со странични канали користат информации за време, потрошувачка на енергија, електромагнетни еманации или дури звук за да откријат тајни информации за криптосистемот.

Временските напади ги користат варијациите во времето при криптографските операции. Поради оптимизација на перформансите, пресметките изведени со криптографски алгоритам често земаат различно време во зависност од влезот и вредноста на тајниот параметар. Ако операциите со приватен клуч на RSA можат да се мерат доволно точно, во некои случаи може да се примени статистичка анализа за да се открие тајниот клуч вклучен во пресметките.

Операцијата во RSA која го вклучува приватниот клуч е модуларна експоненцијација M = C^d mod N, каде што N е RSA модулот, C е текстот што треба да се декриптира или потпише, а d е приватниот клуч. Целта на напаѓачот е да го најде d. Напаѓачот во "timing attack" (напад врз база на време), мора да има таргет систем за да изврши Cd mod N за повеќе претходно избрани вредности на C. Со прецизно мерење на потребното време и анализирање на варијациите во времето, напаѓачот може да го открие приватната клуч d бит по бит сè додека не биде познат целиот експонент. Како што објаснува Kocher, нападот во суштина е проблем на детекција на сигнали. "Сигналот" се состои од варијацијата во времето предизвикана од битот на целниот експонент, додека "шумот" се состои од неточностите во мерењето на времето и случајните флуктуации во времето, особено преку мрежа. Бројот на примероци на времето што се бараат е детерминиран од својствата на сигналот и шумот. Во секој случај, бараниот број на примероци е пропорционален на бројот на битови во експонентот d. Бидејќи има само ограничен број на битови во приватниот експонент d, нападот е компјутерски практичен.

```
x = C
for j = 1 to n
    x = mod(x^2, N)
    if d_j == 1 then
       x = mod(xC, N)
    end if
next j
return x
```

Figure 1: Square and multiply algorithm.

Алгоритмот "Square and multiply" прикажан во Слика 1 е модуларна експоненцијација која се користи во RSA.

Модуларната експоненцијација во RSA се состои од голем број подигнат на голем експонент, што е временски зафатен процес. Едноставниот и ефикасен алгоритам за пресметка на C^d mod N е алгоритамот "Square and multiply", како што е прикажан во Слика 1, каде d = d0d1...dn во бинарен формат, со d0 = 1. На пример, вредноста на експонентот 20 е 10100 во бинарен формат. Без

водечки нули, d0 секогаш е 1. Забележете дека 20 може да се создаде еден бит по еден бит одлево на десно како (0, 1, 10, 101, 1010, 10100) = (0, 1, 2, 5, 10, 20). Исто така, експонентот 20 може да се изгради преку серија на чекори, каде во секој чекор, го дуплираме бројот со поместување на еден бит лево и додаваме 1 ако следниот бинарен бит е 1. За изградба на 20, чекорите се:

1 = 0 * 2 + 1

2 = 1 * 2

5 = 2 * 2 + 1

10 = 5 * 2

20 = 10 * 2.

Сега претпоставете дека сакаме да пресметаме 5^20 mod 33, ефикасен начин би бил да се користи методот на квадрат и множи (square and multiply).

$$5^1 = (5^0)^2 * 5^1 = 5 \bmod 33$$
$$5^2 = (5^1)^2 = 5^2 = 25 \bmod 33$$
$$5^5 = (5^2)^2 * 5^1 = 25^2 * 5 = 3125 = 23 \bmod 33$$
$$5^{10} = (5^5)^2 = 23^2 = 529 = 1 \bmod 33$$
$$5^{20} = (5^{10})^2 = 1^2 = 1 \bmod 33$$

Типичните имплементации на RSA користат Montgomery алгоритмот за извршување на множење и квадратни операции. Со Montgomery алгоритмот, множењата земаат постојано време, независно од големината на факторите. Но, ако претпоставеното резултат од множењето надминува модулусот N, се извршува дополнително одземање, наречено "extra reduction". Во моментот на ова дополнително одземање се предизвикува разлика во времето за различни влезови и се открива информација за тајниот клуч.

Уште еден типичен начин за оптимизација на имплементацијата на RSA е користење на Кинеската теорема на остатоците (Chinese Remainder Theorem - CRT) за извршување на експоненцијацијата. Со CRT, функцијата M = C^d mod N се пресметува така што прво се пресметуваат M1 = C^d1 mod N и M2 = C^d2 mod N, каде d1 и d2 се претпресметани вредности на d. Потоа M1 и M2 се комбинираат за да се добие M. RSA со CRT го прави првобитниот напад на Kocher неефикасен.

2.3 Илустрација на временски напад

Нека C биде порака за дешифрирање, d нека биде приватниот клуч што треба да се открие, означен со d0, d1, ..., dn каде што d0 = 1. Пресметката што треба да се изврши е C^d mod N. Предмет на разгледување е алгоритмот за квадрат и множи (Фигура 1), кој се користи за пресметка на оваа модуларна експоненција, и операцијата mod(x, N) која е имплементирана како :

```
mod(x, N)
if x >= N
   x = x % N
end if
return x
```

Figure 2: Pseudocode for the mod function

Да го разгледаме алгоритмот "Square and multiply" во Фигура 1. Ако dj = 0, тогаш x = mod(x2, N), но ако dj = 1, тогаш се случуваат две операции, x = mod(x2, N) и x = mod(xC, N). Со други зборови, времето на пресметка треба да биде подолго кога dj = 1. Исто така, забележете во Фигура 2 дека модуларното намалување, "mod", се извршува само ако интермедијалниот резултат од множењето е поголем од модулусот N. Нашиот напад искористува две факти и користи две групи на пораки, една за која пресметката на mod(xC, N) би била потребна редукција, а друга за која не, за да се открие било кој бит dj.

Предпоставете дека можеме да го натераме системот да ги дешифрира пораките кои ние ќе ги избереме, што често е можно во реални системи. Почнуваме со напаѓање на d1 со избирање на две пораки Y и Z, каде што Y^3 < N и Z^2 < N < Z^3.

Споредете ги операциите на алгоритмот за квадрат и множи (Фигура 1) врз две пораки Y и Z кога j = 1. Ако d1 е вистински 1, ќе се изврши операцијата x = mod(x · x2, N). Бидејќи Y^3 < N, операцијата "mod" не се случува за пораката Y. Но бидејќи Z^2 < N < Z^3, операцијата "mod" се случува за пораката Z. Со други зборови, времето на извршување на алгоритмот ќе биде подолго само за Z ако d1 = 1. Можеме да го реконструираме битот d1 со помош на овој факт.

Но колку поголема разликата во времето на извршување е значајна? Ќе треба да се потпреме на статистика тука. Наместо да користиме една порака Y, ние избираме група пораки Yi, за i = 0, 1, ..., m-1, каде што Yi^3 < N, и нека yi биде времето потребно за пресметка на Yi^d mod N. Исто така, избираме група пораки Zi, за i = 0, 1, ..., m-1, каде што Zi^2 < N < Zi^3, и нека zi биде времето потребно за пресметка на Zi^d mod N. Ако d1 е фактички 1, сите zi-ња треба да бидат поголеми од yi-њата.

Пресметајќи ја просечната вредност на времето y = (y0 + y1 + ... + ym-1) / m и z = (z0 + z1 + ... + zm-1) / m. Ако z > y + e, каде што e може да се одреди емпирички, заклучуваме дека d1 = 1; инаку d1 = 0. Штом д1 е познат, можеме да нападнеме d2 со сличен процес со избирање на вредности на Y и Z

што задоволуваат различни критериуми. Приватниот клуч d може да се реконструира бит по бит на овој начин.

Убавината на овој напад лежи во неговата едноставност. За да функционира нападот, не ни требаат деталите за имплементацијата на алгоритмот и не ни треба да знаеме како да факторизираме големи броеви - се што ни треба е да знаеме дека експоненцијацијата се извршува со помош на квадрирај-и-множи.

2.4 Едноставна имплементација на временски напад врз RSA во Јава

```
import java.math.BigInteger;
import java.security.SecureRandom;

public class TimingAttackRSA {
    private static final int NUM_SAMPLES = 1000; // Бројот на примероци за
земање време

    public static void main(String[] args) {
        // Генерирање на RSA клучеви
        BigInteger n = generateLargePrime();
        BigInteger e = new BigInteger("65537");
        BigInteger d = e.modInverse(n);

        // Временско мерење на криптоанализата
        long startTime = System.nanoTime();
        for (int i = 0; i < NUM_SAMPLES; i++) {
            // Генерирање на порака и криптирање
            BigInteger message = generateRandomMessage(n);
            BigInteger ciphertext = encrypt(message, e, n);

            // Криптоанализа со мерење на времето
            timingAttack(ciphertext, d, n);
        }
        long endTime = System.nanoTime();
        long totalTime = endTime - startTime;

        System.out.println("Средно време за криптоанализа: " + totalTime /
NUM_SAMPLES + " наносекунди");
    }

    // Генерирање на големо просто бројче
    private static BigInteger generateLargePrime() {
        return BigInteger.probablePrime(1024, new SecureRandom());
    }

    // Генерирање на случајна порака помала од n
    private static BigInteger generateRandomMessage(BigInteger n) {
        SecureRandom random = new SecureRandom();
        BigInteger message;
        do {
            message = new BigInteger(n.bitLength(), random);
        } while (message.compareTo(n) >= 0);
        return message;
```

```
    }

    // Криптирање на пораката со RSA
    private static BigInteger encrypt(BigInteger message, BigInteger e,
BigInteger n) {
        return message.modPow(e, n);
    }

    // Криптоанализа со мерење на времето
    private static void timingAttack(BigInteger ciphertext, BigInteger d,
BigInteger n) {
        long startTime = System.nanoTime();
        ciphertext.modPow(d, n);
        long endTime = System.nanoTime();
        long elapsedTime = endTime - startTime;


    }
}
```

Објаснување:

Прво, генерираме два големи прости броеви n и d, што се користат како RSA клучеви. Со помош на timingAttack функцијата, правиме серија од NUM_SAMPLES примероци за криптоанализа. Во секој примерок, генерираме случајна порака помала од n, ја криптираме со јавниот клуч e и ја извршуваме криптоанализата со мерење на времето. Конечно, го прикажуваме просечното време за криптоанализата на стандарден излез.

2.5 Можни одбрани

Најшироко прифатен метод е RSA заслепувањето. Со заслепувањето на RSA, случајноста се воведува во пресметките на RSA за да се направат информациите за тајмингот неупотребливи. Пред да го дешифрираме шифрениот текст C, прво пресметуваме X = r^eC mod N, каде r е случајна вредност и е е јавен експонент. Ние го дешифрираме X како и обично, т.е., пресметуваме X^d mod N =r^edC^d mod N = r C^d mod N повторно со Ојлеровата теорема. Потоа го множиме излезот со r^-1 за да добиеме C^d mod N што е чист текст што го сакаме. Бидејќи за секоја порака се користи различно r, оригиналната порака се менува на случаен начин пред операцијата за степенување. Така, заслепувањето го спречува напаѓачот да внесе познат влез во функцијата за степенување и да ги користи добиените информации за времето за да го открие клучот. Заслепувањето носи мала казна за изведба во опсег од 2% до 10%.

Заклучок

Во овој проект се даде детален опис на криптографскиот алгоритам RSA, и опсежно се прикажа временскиот напад врз него. Исто така во проектот се изведе и кратка импровизација на временскиот напад врз RSA во програмскиот јазик JAVA.

Временските напади илустрираат дека напаѓачите не играат неопходно според претпоставените правила и дека секогаш ќе се нападне најслабата точка во системот.Временските напади претставуваат сериозна загриженост за безбедноста на криптографските системи, вклучувајќи го и алгоритмот RSA. Преку воведување на техники и контроли за одбрана, може да се зголеми отпорноста на RSA алгоритмот против временските напади.

За да се заштити RSA алгоритмот од временски напади, е неопходно да се применат техники за одбрана, како што се корекции на временските разлики, случајни доцнења и контроли на рамнотежната обработка. Исто така, редизајнирање на самите алгоритми или користење на други криптографски методи може да биде потребно за да се избегне ризикот од временски напади.

Референци

https://paulkocher.com/doc/TimingAttacks.pdf

https://kastner.ucsd.edu/ryan/wp-content/uploads/sites/5/2014/03/admin/RSA-timing-attack.pdf

https://en.wikipedia.org/wiki/Timing_attack

https://www.cs.sjsu.edu/faculty/stamp/students/article.html

Изработи:

Станковска Петрина, 201073