Hello, World!

Асиметрична криптографија и RSA

Петар Величковић

Циљ *криптографског алгоритма* је да омогући безбедну комуникацију између два ентитета, која ћемо у даљем тексту звати А (Алиса) и Б (Боб). Ово се најчешће постиже претварањем сирових порука, P, између А и Б у шифроване, C (и обратно), тако да, уколико било који прислушкивач, кога ћемо у даљем тексту звати Е (Ева), пресретне њихову комуникацију, није у могућности да реконструише оригиналне поруке из шифрованих у разумном временском интервалу. Поступак претварања P у C зовемо *енкрипцијом*, док поступак реконструисања P из C називамо *декрипцијом*. Углавном, ове функције ће бити параметризоване неким кључем K; пишемо $C = Enc_K(P)$, $P = Dec_K(C)$. Претпостављаћемо да су P, C и K цели бројеви.

Основна неопходна својства криптографског алгоритма је формализовао Керкхофс (Kerckhoffs) 1883. године. Вероватно најважније од тих својстава је да **не сме** да се рачуна на *тајновитост алгоритма*, тј. при анализи безбедносних својстава неког криптографског алгоритма увек претпостављамо да Е зна његову имплементацију! Стога, безбедност мора **комплетно** да се ослања на *тајновитост кључа* K!

У овом тексту ћемо представити један од тренутно најкоришћенијих алгоритама у криптографији (RSA). Показаћемо како се може директно применити за шифровање велике количине података на Интернету.

Симетрична криптографија и криптографске „игре“

Укратко ћемо размотрити ситуацију у којој је потребно подржати безбедну комуникацију између А и Б користећи тајни кључ K који је познат једино њима. Ова ситуација се назива још и *симетричном криптографијом*, зато што А и Б имају идентичне информације. За тестирање безбедности неког алгоритма симетричне криптографије, дефинисаног преко функција (Enc, Dec), користи се „игра“ са следећим правилима:

Најпре се генерише насумичан кључ K, дужине l.

- Затим, „непријатељ“ има право да позива функције Enc_K и Dec_K на произвољно много улаза.

- Непријатељ онда исписује две поруке, P_1 и P_2. Насумице се бира једна од те две поруке, означимо је са P', и шифрује се, тј. непријатељу се враћа $C' = Enc_K(P')$.
- Непријатељ и даље има право да позива Enc_K и Dec_K на произвољно много улаза, али *не сме* да тражи $Dec_K(C')$!

- Коначно, непријатељ исписује која је, по његовом мишљењу, порука P'.

Алгоритам се сматра безбедним уколико непријатељ није у могућности да након полиномно много упита функција Enc_K и Dec_K добије довољно информација о томе да ли је $P' = P_1$ или $P' = P_2$. Формалније, уколико непријатељ исписује поруку P'', потребно је да важи $\mathbf{P}\,(P' = P'') < \frac{1}{2} + \epsilon(l)$, где је $\epsilon(l)$ *„занемарљиво мала“* функција. Конкретно, када $l \to +\infty$, $\epsilon(l)$ мора да брже тежи нули од $\frac{1}{poly(l)}$, где је $poly(l)$ *било који* полином. Ово нам омогућава да ефикасно повећамо број корака који непријатељ мора да направи, само повећавајући дужину кључа l.

Једна јако важна последица овакве формулације игре је да функција Enc_K **не може бити детерминистичка** (тј. не сме увек за исти улаз да да исти излаз)! Доказ ове последице вам остављамо као један од задатака на крају чланка.

За потребе овог текста, сматраћемо да нам је овакав алгоритам, тј. пар функција Enc_K и Dec_K, са жељеним својствима, доступан као „црна кутија“. Начин изградње таквог алгоритма захтева дубљи увод у криптографске примитиве (попут AES, CTR, CMAC итд.) и стога му нећемо посвећивати додатну пажњу овде.

Велики проблем симетричне криптографије је ограничена **скалабилност**. Уколико имамо n учесника у систему и сваки пар би потенцијално желео да безбедно комуницира, ово би захтевало да се на систематичан и безбедан начин додели $O(n^2)$ кључева. То се додатно компликује уколико учесници могу произвољно да улазе и излазе из система. Ово убрзо постаје проблематично са растом n и практично је немогуће за масивне системе попут онаквих какви се срећу на Интернету. Један од начина на који се овај проблем може превазићи је прелаз на *асиметричну криптографију*.

Асиметрична криптографија и безбедне хеш функције

Претпоставимо да ентитет А жели да прима шифроване поруке од неког (потенцијално великог) броја других ентитета, тако да само А и пошиљалац могу да знају њихов садржај. Алгоритам *асиметричне криптографије* то омогућава, тако што, уместо једног кључа, постоје два: *јавни кључ* (PK) и *тајни кључ* (SK). Функције за шифровање и дешифровање сада користе одвојене кључеве: $C = Enc_{PK}(P)$ и $P = Dec_{SK}(C)$. Ентитет А задржава тајни кључ за себе, док се јавни кључ **објављује свима**. Овакав алгоритам омогућава пријем безбедних порука од *великог броја ентитета* (зато што је кључ потребан за шифровање јавно доступан) користећи *само један пар кључева*. Стога се изузетно често примењује унутар безбедносних протокола на Интернету.

И за овај сценарио можемо дефинисати еквивалентне криптографске „игре“ којима се доказује безбедност алгоритма; једина разлика је у томе што сада непријатељ добија и приступ јавном кључу PK.

У остатку текста ћемо демонстрирати како, користећи популарни RSA (Rivest-Shamir-Adleman) алгоритам, можемо дефинисати безбедан протокол за асиметричну криптографију. Да би ово било могуће, такође ћемо претпоставити да имамо на располагању и *безбедну хеш функцију* h, детерминистичку функцију која има фиксну максималну дужину излаза, такву да је **јако тешко** (сложеност $\approx 2^{80}$ операција) пронаћи два улаза x и x', тако да важи $x \neq x' \wedge h(x) = h(x')$. Теоретски, постојање овакве функције није могуће; вредности x и x' морају постојати, и „тривијални“ непријатељ би само исписао један такав пар вредности). Међутим, постоје функције које већ годинама успешно одолевају практичним нападима оваквог типа (нпр. SHA-3), и стога се за наше потребе могу сматрати безбедним.

RSA алгоритам

Најпре је потребно одабрати два различита, велика, проста броја p и q (углавном величине око 1024 бита). Размотримо број $n = pq$. Пошто је n производ два проста броја, број узајамно простих бројева са n, а мањих од n, једнак је $\phi(n) = \phi(pq) = \phi(p)\phi(q) = (p-1)(q-1)$. (У другом кораку смо користили чињеницу да су p и q узајамно прости.)

По Ојлеровој теореми, знамо да је

$$\text{nzd}(a, n) = 1 \Leftrightarrow a^{\phi(n)} \equiv 1 \pmod{n}.$$

Последица овога је да унутар скупа Z_n важи $a^x \equiv a^{x \bmod \phi(n)} \bmod n$. Из тога, даље, следи да ако одаберемо пар целих бројева (e, d) таквих да је $ed \equiv 1 \pmod{\phi(n)}$, онда важи $a^{ed \bmod \phi(n)} = a^1 = a$. (Један од задатака је посвећен начину на који се бројеви e и d бирају.) Тада можемо e и d употребити као јавни и тајни кључ, респективно, и потом (де)шифровати степеновањем поруке, односно шифрата. Конкретно:

- $PK = (n, e), SK = (n, d)$.
- $Enc_{PK}(P) = P^e \bmod n$.
- $Dec_{SK}(C) = C^d \bmod n$.

Очигледно, важи

$$Dec_{SK}(Enc_{PK}(P)) \equiv P^{ed} \equiv P^{ed \bmod \phi(n)} \equiv P^1 \equiv P \pmod{n}.$$

Стога, овај алгоритам омогућава примаоцу који поседује тајни кључ да коректно дешифрује све поруке шифроване јавним кључем. Дешифровање изоловане поруке (у одсуству других, претходно послатих, порука) без тајног кључа, тренутно не може да се уради „лакше“ од факторисања броја n. А то је проблем за који тренутно не постоји класичан алгоритам полиномне сложености. Додуше, постоји *квантни* алгоритам полиномне сложености, али квантни рачунари су још увек далеко од ефикасног факторисања. Звучи невероватно, али највећи број који су тим алгоритмом до сада факторисали је 21.

Наивна примена RSA алгоритма у овом облику, нажалост, није довољно безбедна да би ”победила” непријатеља у наведеној криптографској игри. (Конкретни напади су остављени за наградни задатак.) То је углавном зато што поруке нису *изоловане*. Овај проблем можемо заобићи тако што употребимо RSA само за преношење *новог кључа*, који онда користимо за шифровање саме поруке симетричном криптографијом! Пошто RSA ради са порукама унутар скупа Z_n, а симетричан алгоритам може користити кључеве из различитог скупа, кључ ће заправо бити резултат примене безбедне хеш функције на послату вредност. Да би избегли претходно наведе-

не проблеме са изолацијом, послата вредност ће бити насумично генерисана. Свака следећа комуникација насумично генерише нови кључ. Конкретно:

- Претпостављамо да имамо безбедан систем за симетричну криптографију, тј. функције Enc_K и Dec_K, као и безбедну хеш функцију h.

- Претпостављамо да смо већ генерисали јавни и тајни RSA кључ, $PK = (n, e)$ и $SK = (n, d)$.

- Пошиљалац при слању поруке P најпре насумично генерише вредност x. Ова вредност се шаље користећи RSA. У исто време се и користи за шифровање P симетричним системом користећи хеширан x као кључ $K = h(x)$! Комплетна послата порука је тада $(x^e \bmod n,\ Enc_{h(x)}(P))$.

- Прималац најпре дешифрује вредност x користећи тајни кључ $(x = (x^e)^d \bmod n)$, а затим сазнаје кључ $K = h(x)$ и користи га да дешифрује поруку $P = Dec_K(Enc_K(P))$.

Уколико бисмо заменили хеш функцију потпуно насумичном функцијом, оваква конструкција би гарантовано имала жељена безбедносна својства. То значи да, уколико хеш функција у било ком моменту буде пробијена, проблем се може лако решити преласком на „јачу“ хеш функцију.

Задаци

1. Демонстрирајте једноставан напад којим непријатељ може увек да победи у криптографској „игри“ против *детерминистичког* алгоритма симетричне криптографије, тј. алгоритма у коме $Enc_K(P)$ увек даје исти резултат за исте вредности P и K.

2. Практичне имплементације RSA алгоритма углавном постављају параметар e на вредности попут 3, 17 или $2^{16} + 1$. Објасните како овакве и њима сличне вредности омогућавају брзо шифровање порука. Да ли бисмо онда могли да поставимо d на неку од ових вредности да бисмо убрзали дешифровање?

3. Практичне имплементације RSA алгоритма често укључују и факторизацију броја $n = pq$ у тајни кључ, тј. $SK = (p, q, d)$. Објасните како овакав тајни кључ може омогућити брже дешифровање.

4. Важно је запамтити: *Сви, ӣа и најбољи, криӣӣоӣрафски ӣроӣоколи врло лако ӣоклекну уколико се лоше имӣлеменӣирају!* Скорашњи пример овога је *heartbleed*, грешка у имплементацији OpenSSL протокола којим се шифрују информације на Интернету, којом је погођено око 500.000 сервера. Унутар овог протокола, периодично се проверава да ли је веза између ентитета и даље активна тако што се шаљу „heartbeat“ поруке. Пошиљалац шаље серверу неки стринг S и његову дужину L. Због ограничења програмског језика C, дужина се не чува унутар самог стринга. Сервер је у обавези да врати тај исти стринг пошиљаоцу. OpenSSL није проверавао да ли важи $|S| = L$! Објасните како овакав пропуст може омогућити пошиљаоцу директан и неопажен приступ садржају радне меморије сервера у којој се могу налазити тајни кључеви, лозинке, итд!

Статијата прв пат е објавена во списанието Тангента на ДМ на Србија во 2017/18 година