تمرین اول
پیرحسینلو
۴۰۱۴۴۳۰۲۹

۱-
الف)
کلید k بیت است. ابتدا تمام A ها را با امتحان کردن تمام کلیدها به دست می‌آوریم:

$A = E(P, K_1)$
Time: $t1 = O(2^k)$

حال تمام A ها را مرتب می‌کنیم.

Time: $t2 = O(2^k \times \log(2^k)) = O(k \times 2^k)$

حال تمام B ها را به دست می‌آوریم:

$B = D(C, K_2)$
Time: $t3 = O(2^k)$

حال به ازای هر B، در جدول A بررسی می‌کنیم تا ببینیم وجود دارد یا خیر. چون جدول A مرتب است، از Binary Search استفاده می‌کنیم:

For each B: $O(2^k)$
    Perform Binary Search: $O(\log(2^k)) = O(k)$
Time: $t4 = O(2^k \times \log(2^k)) = O(k \times 2^k)$

Total Time: $t1 + t2 + t4 = O(2^k + k \times 2^k + k \times 2^k) = O(k \times 2^{k+1})$

زمان مورد نیاز برای شکستن رمزنگاری واحد در بدترین حالت با امتحان کردن تمام کلیدها:

Time: $O(2^k)$

$k \times 2^{k+1} / 2^k = 2k$

در نتیجه، 2k برابر نسبت به حالت رمزنگاری واحد زمان نیاز است تا رمزنگاری دوگانه شکسته شود.

ب)
ابتدا تمام A ها را با امتحان کردن تمام کلیدها به دست می‌آوریم:

$A = E(P, K_1)$
Time: $t1 = O(2^k)$

حال تمام B ها را به دست می‌آوریم:

$B = D(C, K_3)$
Time: $t2 = O(2^k)$

حال به ازای هر جفت A و B، می‌خواهیم معادله‌ی مقابل را حل کنیم:

$B = E(A, K_2)$

این قسمت مانند قسمت الف) حل می‌شود.

$\text{Total Time} = O(2^k + 2^k + k \times 2^k \times 2^k) = O(k \times 2^{2 \times k})$

ج)

$\text{Total time} = O(k \times 2^{k \times (n - 1)})$

۲-

اگر N_1 و N_2 نسبتا اولا نباشند، یعنی coprime نباشند، یک A وجود دارد به طوری که

$N_1 = A \times B_1$
$N_2 = A \times B_2$

در واقع A همان بزرگترین مقسوم علیه مشترک آنهاست (gcd). با توجه به اینکه فاکتورهای مشترک در A نهادینه می‌شوند، B_1 و B_2 نسبت به هم اول هستند در غیر این صورت فاکتورهای مشترک در gcd ظاهر می‌شدند. حال موارد زیر را در نظر بگیرید:

1. $N_1 = gcd(N_1, N_2) \times B_1$ (با توجه به موارد ذکر شده)
2. $N_2 = gcd(N_1, N_2) \times B_2$ (با توجه به موارد ذکر شده)
3. $gcd(B_1, B_2) = 1$ (با توجه به موارد ذکر شده)
4. $N_1 = P_1 \times Q_1$ (طبق RSA)
5. $N_2 = P_2 \times Q_2$ (طبق RSA)
6. $gcd(P_1, Q_1) = 1$ (طبق RSA)
7. $gcd(P_2, Q_2) = 1$ (طبق RSA)

با توجه به این موارد می‌توان نتیجه گرفت:

$P_1 = P_2 = gcd(N_1, N_2)$
$Q_1 = B_1$
$Q_2 = B_2$
$\Phi(N_1) = (gcd(N_1, N_2) - 1) \times ((N_1 / gcd(N_1, N_2)) - 1)$
$\Phi(N_2) = (gcd(N_1, N_2) - 1) \times ((N_2 / gcd(N_1, N_2)) - 1)$

از آنجا که محاسبه‌ی gcd در زمان لگاریتمی امکان‌پذیر است، Φ در زمان کوتاهی محاسبه می‌شود.

۳- الف)

$\boxed{gcd(N, r) = 1}$

می‌خواهیم plaintext مربوط به c را به دست آوریم. ابتدا یک r انتخاب می‌کنیم. سپس داریم:

$c'' = c \times r^e \bmod N$

حال c'' را به سیستم قربانی می‌دهیم تا رمزگشایی کند. سیستم قربانی مراحل زیر را طی می‌کند:

$(c'')^d \bmod N = (c \times r^e)^d \bmod N = (c^d \times r^{ed}) \bmod N$

از طرفی می‌دانیم:

$e \times d \equiv 1 \bmod \Phi(N)$
$e \times d \equiv 1 \bmod (p - 1) \times (q - 1)$
$e \times d = 1 + k \times (p - 1) \times (q - 1)$ for some k

ادامه‌ی کار سیستم قربانی:

$(c^d \times r^{e \times d}) \bmod N = (c^d \times r^{(1 + k \times (p - 1) \times (q - 1))}) \bmod N = (c^d \times r \times r^{(k \times (p - 1) \times (q - 1))}) \bmod N = (c^d \times r \times (r^{q - 1})^{k \times (p - 1)}) \bmod N$

طبق Fermet's little theroem، داریم:

$r^{q - 1} \bmod N = 1$

در نتیجه:

$(c^d \times r \times (r^{q - 1})^{k \times (p - 1)}) \bmod N = (c^d \times r \times 1) \bmod N = (c^d \times r) \bmod N$

سیستم قربانی نتیجه‌ی خط قبل را برای attacker ارسال می‌کند. attacker نیز با تقسیم بر r، به plaintext متن رمزشده‌ی اولیه دست می‌یابد.

$(c^d \times r) \bmod N / r = (c^d) \bmod N$

خط قبل در اصل همان plaintext متناظر c را ایجاد می‌کند.

ب)

$e \times d \equiv 1 \bmod (p - 1) \times (q - 1)$
$e \times d = 1 + k \times (p - 1) \times (q - 1)$ for some k
$e \times d - 1 = k \times \Phi(N)$
$3 \times d - 1 = k \times \Phi(N) = u_1^{v1} \times u_1^{v2} \times \dots u_w^{vw}$ تجزیه به فاکتورها

از آنجا که p و q اعداد اول بزرگ هستند، پس فرد هستند (تنها عدد اول زوج، ۲ است.). بنابراین، p - 1 و q - 1 زوج هستند. در نتیجه:

$3 \times d - 1 = k \times \Phi(N) = 2^{v1} \times u_2^{v2} \times \dots u_w^{vw}$ تجزیه به فاکتورها

حال می‌توان گفت که تمام فاکتورها فرد هستند.

(من این تمرین رو تو نت جوابش رو دیدم. از اینجا به بعد رو دلیلش رو نفهمیدم. باید بیشتر فکر کنم ببینم چرا اینطوری میشه.)

حال برای هر u_i، عبارت زیر را حساب می‌کنیم:

$x = (N + 1 - pi) / 3$

اگر x یک عدد صحیح شد، می‌توان با استفاده از آن، فاکتورهای N را حساب کرد.

۴-

الف) احتمال ورود پس از n بار برابر است با احتمال شکست در n بار سپس ورود در دفعه‌ی بعد. بنابراین:

$p(sucess) = 1 - (1 - 1/2^k)^n$
$(1 - 1/2^k)^n = 1 - p(sucess)$

$n \times \log(1 - 1/2^k) = \log(1 - p(sucess))$
$n = \log(1 - p(sucess)) / \log(1 - 1/2^k)$
$k = 20, p(sucess) = 0.0006$
$\text{---->} \quad n \simeq \log(1 - 0.0006) / \log(1 - 1/2^{20}) = 630$

حداقل تعداد حدس‌ها باید برابر 630 باشد.

ب)
احتمال یکسان شدن hash دو کاربر برابر است با:

$(1/2^k) \times (1/2^k) = (1/2^{2k})$

تعداد جفت افراد:

$n \times (n - 1) / 2$
$p(collision) = (n^2 - n) / 2^{2k + 1}$

با جایگذاری اعداد در فرمول، حداکثر تعداد افراد به دست می‌آید.

۵-
الف) به نظرم باید فضا به قدری بزرگ باشد که با استفاده از birthday attack نیز نتوان سندی پیدا کرد که hash آن برابر hash یکی از اسناد فعلی باشد. فکر میکنم ۲۵۶ بیت مناسب باشد.

ب)[1]
فرض کنید داریم:

$s_1 = m_1^d \bmod N$
$s_2 = m_2^d \bmod N$

حال m_3 را به شکل زیر حساب می‌کنیم:

$m_3 = m_1 \times m_2$

حال داریم:

$m_3^d \bmod N = (m_1 \times m_2)^d \bmod N = (m_1^d \times m_2^d) \bmod N = ((m_1^d \bmod N) \times (m_2^d \bmod N)) \bmod N = (s_1 \times s_2) \bmod N$

attacker طبعا به s_1 ،N و s_2 دسترسی دارد. پس می‌توان بدون کلید خصوصی فرستنده، برای پیام $m_1 \times m_2$ یک امضا بسازد و آن را از طرف فرستنده ارسال کند.

[1] https://crypto.stackexchange.com/questions/9896/how-does-rsa-signature-verification-work