Kryptoanaliza stosowana 2025

Lista zadań nr 6b: bezpieczeństwo bitów RSA, ataki z wyroczniami *paddingu*

Na zajęcia 1 grudnia 2025

Zadanie 1 (2 pkt). Zaimplementuj atak na RSA z wyroczniami half i parity omówiony na wykładzie. Zademonstruj jego działanie dla 1024-bitowego klucza.

Zadanie 2 (2 pkt). Do niniejszego pliku PDF zostało dołączone archiwum `padding.tar.xz` zawierające kilka modułów i programów w Pythonie implementujących algorytm RSA i padding zdefiniowany w PKCS#1 v. 1.5. W głównym katalogu znajduje się zalążek programu w Pythonie o nazwie `zadanie1`, który ma symulować atak Bleichenbachera. Program wczytuje klucz RSA i przechwycony szyfrogram. Wyrocznia jest uproszczona, odpowiada jedynie na pytanie, czy odszyfrowany ciąg ma właściwy prefiks `0x0002` (tj. czy $2B \le m \le 3B - 1$). Uzupełnij ten program i, odpytując inteligentnie wyrocznię, złam szyfrogram. Maleństwo potrzebowało 6016 zapytań, a atak trwał kilka sekund. Możesz wyrocznię zmienić na prawdziwą, ale wówczas trzeba będzie poczekać minutę lub dwie (28411 zapytań).

Zadanie 3 (1+1 pkt). W podkatalogu `bleichenbacher/` archiwum `padding.tar.xz` znajdują się też pliki `RSAkey2` i `cipher2.txt`. Teraz klucz ma już praktyczną długość 1024 bitów. Złam ten szyfrogram korzystając z uproszczonej (1 pkt) lub prawdziwej (dodatkowy punkt) wyroczni. Na ten drugi punkt właściwie zarobisz nie Ty, tylko Twój komputer! (Maleństwo dało radę, ale się zgrzało — 149077 zapytań, 20 minut.)

Zadanie 4 (1 pkt za każdą pracę). Przeczytaj poniższe prace:

- Vlastimil Klíma, Ondřej Pokorný, Tomáš Rosa, Attacking RSA-Based Sessions in SSL/TLS, *Cryptographic Hardware and Embedded Systems – CHES 2003*, LNCS 2779:426–440.
- Romain Bardou, Riccardo Focardi, Yusuke Kawamoto, Lorenzo Simionato, Graham Steel, Joe-Kai Tsay, Efficient Padding Oracle Attacks on Cryptographic Hardware, *Advances in Cryptology – CRYPTO 2012*, LNCS 7417:608–625.
- Tibor Jager, Sebastian Schinzel, Juraj Somorovsky, Bleichenbacher's Attack Strikes again: Breaking PKCS#1 v1.5 in XML Encryption, *17th European Symposium on Research in Computer Security — ESORICS 2012*, LNCS 7459:752–769.
- Tibor Jager, Jörg Schwenk, Juraj Somorovsky, On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption, *21st Conference on Computer and Communications Security — CCS'15*, pp. 1185–1196.
- Hanno Böck, Juraj Somorovsky, Craig Young, Return Of Bleichenbacher's Oracle Threat (ROBOT), *27th USENIX Security Symposium*, 2018. (Do zreferowania podczas zajęć możesz użyć ich slajdów z prezentacji na konferencji.)
- Eyal Ronen, Robert Gillham, Daniel Genkin, Adi Shamir, David Wong, Yuval Yarom, The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations, *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 435–452.

i przygotuj się do ich zreferowania podczas zajęć.

Zadanie 5 (2 pkt). Rozważmy szyfr RSA z modułem n i wykładnikami e i d (d jest znane tylko wyroczni) oraz przechwycony szyfrogram c_0 odpowiadający nieznanemu tekstowi jawnemu m_0. Niech b_0 będzie takie, że $b_0 \le \lfloor n/2 \rfloor$ oraz niech będzie *dostatecznie duże* (zaraz wyjaśnimy, jak bardzo). Niech $I = [0, b_0]$. Rozważmy wyrocznię, która dla danego szyfrogramu $c \in [0, n)$ odpowiada, czy odszyfrowany tekst jawny $m = c^d \bmod n \in I$. Mamy wówczas następujący atak, który pozwala na odszyfrowanie nieznanej wiadomości m_0 za pomocą $O(\log_2 n)$ zapytań do wyroczni.

Możemy założyć, że $m_0 \perp n$. W przeciwnym razie możemy odszyfrować wszystko (dlaczego)?

Na początek należy wylosować takie dwie liczby s_i $(i = 1, 2)$, że $c_i = (s_i^e c_0) \bmod n$ spełniają wyrocznię (im mniejsze jest b_0, tym więcej prób trzeba wykonać). Mamy wówczas $m_i = s_i m_0 \bmod n \le b_0$. Zapytajmy wyrocznię o $c(s_2 - s_1)^e \bmod n$. Odpowiedź będzie pozytywna wtedy i tylko wtedy, gdy $(m_2 - m_1) \bmod n \le b_0$, a skoro $b_0 \le \lfloor n/2 \rfloor$, to będzie tak wtw, gdy $m_2 > m_1$. Umiemy więc porównywać teksty jawne, mimo że ich nie znamy! Przypuśćmy, że odpowiedź wyroczni była pozytywna (w przeciwnym razie zamieńmy m_i miejscami).

Za pomocą wyszukiwania binarnego możemy znaleźć największe takie k, że $m_2 > km_1$, a zatem $m_2 = km_1 + r$, gdzie $0 \le r < m_1$. Umiemy więc także dzielić nieznane teksty jawne z resztą $r = m_2 - km_1 = (s_2 - ks_1)m \bmod n$.

Umiejętność porównywania liczb oraz ich dzielenia z resztą wystarcza do zaimplementowania algorytmu Euklidesa wyznaczającego największy wspólny podzielnik $t = \gcd(m_1, m_2)$. Mamy wtedy $t = (jm) \bmod n$, przy czym j znamy (t oczywiście nie znamy, ale mamy go w zaszyfrowanej postaci). Jeśli $m_1 \perp m_2$, to $t = 1$, a wtedy $m = j^{-1} \bmod n$. Jeśli nie, to mamy pecha i powtarzamy całą procedurę. Prawdopodobieństwo, że dwie losowe liczby wybrane z przedziału $[0, n)$ są względnie pierwsze dąży wraz z n do $6/\pi^2 \approx 0.6$. Wielu powtórek nie będzie więc potrzeba. Zaimplementuj ten atak.

Wyrocznię z powyższego zadania można uogólnić tak, by odpowiadała na pytanie, czy $m \in [a_0, b_0]$, tak jak w ataku Bleichenbachera.[1] Mimo to jednak nie da się go wykorzystać do atakowania paddingu PKCS#1. Czemu?

Zadanie 6 (2 pkt). Rozważmy teraz wyrocznię parity podobną do tej z zadania 1, teraz jednak $\mathrm{parity}(n) = 1$, jeśli $2 \mid n$ oraz $\mathrm{parity}(n) = 0$ w p.p. Niech

$$\begin{aligned} s_0 &= 1, \\ s_i &= (2^{-1} s_{i-1}) \bmod n, \quad \text{dla } i > 0, \\ u_0 &= 0, \\ u_i &= \frac{1}{2}\left(u_{i-1} + \mathrm{parity}(s_i u_{i-1} \bmod n)\right), \quad \text{dla } i > 0, \end{aligned}$$

przy czym 2^{-1} jest elementem odwrotnym do 2 modulo n, więc $s_i \in \mathbb{Z}_n$, zaś $u_i \in \mathbb{Q}$ i operacje arytmetyczne w ostatnim równaniu są wykonywane na ułamkach zwykłych. Udowodnij, że

$$|(s_i m) \bmod n - u_i n| \;<\; \frac{n}{2^i}.$$

Połóżmy $r = \lfloor \log_2 n \rfloor + 1$. Wtedy $|(s_i m) \bmod n - u_i n| < 1/2$, a więc $(s_r m) \bmod n = \lfloor u_r n + 1/2 \rfloor$, a stąd $m = (s_r^{-1} \lfloor u_r n + 1/2 \rfloor) \bmod n$. Zaimplementuj ten atak, zwany *aproksymacją wymierną*. Porównaj go z bisekcją z zadania 1.

Zadanie 7 (2 pkt). Przypuśćmy, że n jest nieparzystą liczbą złożoną, a p jest jej właściwym dzielnikiem. Niech $p - 1 = p_1^{j_1} \dots p_k^{j_k}$ będzie rozkładem liczby $p-1$ na czynniki pierwsze (dwójka występuje w nim w co najmniej pierwszej potędze). Niech $B \ge \max\{p_i^{j_i}\}_{i=1}^k$. Wtedy każde $p_i^{j_i}$ występuje w iloczynie $B!$, a więc $p - 1 \mid B!$, tj. $B! = (p-1)t$ dla pewnego t. Z Małego Twierdzenia Fermata mamy więc $2^{B!} = 2^{(p-1)t} \equiv 1 \pmod p$.

Niech $a = 2^{B!} \bmod n$. Skoro $p \mid n$, to $a \equiv 2^{B!} \pmod p$, a zatem $a \equiv 1 \pmod p$, czyli $p \mid (a-1)$, a skoro także $p \mid n$, to $p \mid \gcd(a-1, n)$. Liczba $d = \gcd(a-1, n) > 1$ jest dzielnikiem n. Jeśli $d \ne n$, to d jest właściwym dzielnikiem n. Jest to jeden z najprostszych wariantów tzw. ataku $p-1$ Pollarda.[2]

Wszystkie obliczenia prowadzimy modulo n, więc dla rozsądnych wartości n, np. 1024-bitowych i rozsądnych wartości B, np. rzędu 10^6, obliczenie a nie jest zbyt kosztowne. Oczywiście nie wyliczamy najpierw $B!$ (bo np. dla $B = 10^6$ liczba $B!$ ma 12 815 519 bitów oraz 5 565 709 cyfr dziesiętnych!), tylko liczymy potęgę $2^{B!} \bmod n$ inkrementacyjnie: $a_1 = 2$, $a_k = a_{k-1}^k \bmod n$, dla $k = 2, \dots, B$.

Aby jednak atak zadziałał, *wszystkie* czynniki $p_i^{j_i}$ liczby $p - 1$ muszą być dosyć małe (powiedzmy — jak dla Maleństwa — mniejsze niż $B = 10^6$), inaczej obliczenie wartości a_k trwałoby zbyt długo. Z drugiej strony, jeśli wyliczymy a_k dla zbyt dużego k tak, że k będzie ograniczać czynniki zarówno $p-1$, jak i $q-1$ (dla $n = pq$), to otrzymamy $d = n$.[3] Dlatego podczas wyliczania kolejnych wartości a_k należy co jakiś czas sprawdzać, czy $\gcd(a_k - 1, n) > 1$. Nie warto tego robić w każdej iteracji, gdyż obliczenie gcd kosztuje (wystarczy pamiętać wartości k i a_k z ostatniej iteracji, w której sprawdzaliśmy gcd). Obliczenia można przyspieszać na wiele sposobów. Na przykład zamiast liczyć $a^{k!} \bmod n$ możemy liczyć $a^{\prod_{i=1}^m p_i^{j_i}} \bmod n$, gdzie $\{p_i\}_{i=1}^m$ jest ciągiem wszystkich m początkowych liczb pierwszych, zaś j_i to ciąg (odpowiednio) dobranych wykładników.

Zaimplementuj ten atak, a następnie złam poniższy 1024-bitowy klucz RSA i odczytaj wiadomość.[4] Rozłóż $p-1$ na czynniki pierwsze i zobacz, że wszystkie są małe. Wyznacz minimalne B w tym przypadku. Z drugiej strony zauważ, że złamany klucz prywatny (p, q) wygląda całkiem przyzwoicie i aż trudno uwierzyć, że jest tak podatny!

---

[1] Zob. Michael Ben-Or, Benny Chor, Adi Shamir, On the cryptographic security of single RSA bits, *15th ACM Symp. Theory Comput. STOC 1983*, pp. 421–430.

[2] Zob.: Douglas R. Stinson, Maura B. Paterson, *Cryptography. Theory and Practice*, 4th ed., CRC Press, 2019, podrozdział 6.6.1, str. 212–213.

[3] Może też się zdarzyć, że $d = n$ nawet jeśli tak nie jest. Wówczas można obliczenie powtórzyć dla innego a_1.

[4] Klucz publiczny i szyfrogram znajdują się w pliku `zadanie_7.txt` dołączonym do niniejszego dokumentu PDF.

Dawniej sugerowano, by liczby pierwsze dla RSA generować następująco: losujemy dostatecznie wielką liczbę pierwszą r (np. 256-bitową) i obliczamy $p = kr + 1$ dla dostatecznie wielkiego losowego k tak, by p miała odpowiednią liczbę bitów (np. 512). Następnie sprawdzamy, czy p jest pierwsza. Jeśli nie, czynność powtarzamy. Można pójść nawet dalej i generować jedynie *silne* liczby pierwsze, np. algorytmem Gordona. W praktyce się jednak tak nie robi.[5]

Istnieją różne uogólnienia powyższego ataku. Np. użycie grup eliptycznych (atak Lenstry) daje jeden z najlepszych znanych algorytmów faktoryzacji. Istnieje także podobny atak $p+1$ Williamsa. Katalog wszystkich znanych ataków na RSA (z których większość, to algorytmy faktoryzacji $n = pq$ dla pseudopierwszych liczb p i q wygenerowanych standardową metodą) pozwala oszacować jego kryptograficzną odporność. Na podstawie znanych ataków wyliczono np., że poprawnie wygenerowany klucz 3072-bitowy ma odporność około 128 bitów (złamanie wymaga rzędu 2^{128} kroków obliczeń), a dla odporności rzędu 256 bitów potrzeba klucza rozmiaru rzędu 13 000 bitów. Dobre klucze 1024-bitowe mają około 80 bitów odporności, zatem fakt, że Maleństwo potrafi złamać poniższy klucz w kilkadziesiąt sekund jest wynikiem jedynie tego, że specjalnie wygenerowałem klucz podatny na atak $p - 1$ Pollarda.[6]

$$
\begin{aligned}
n &= 1675067575290949485555011767611678159426379477316211099170663509027389234068435 93\\
&\ 168535386909114107878797517463223419462549477258769908565179009666118709926653 24\\
&\ 856899303144991883036177828611571774630629944450502748538023984324312179468002 84\\
&\ 6095863615913893987848517084807708715419213073080529903198571649689 41\\
e &= 65537\\
c &= 1631659431009204767679121719738698833960069780874404586673616774020992691035939 1\\
&\ 000070593080123050995385596343233461179385837806892945740083249710861798972383 77\\
&\ 949785898639228651379726486855204198248636544410247645386268621141552808794964 83\\
&\ 2602495776817343512156721699497938384189078979116114715816715776132 52
\end{aligned}
$$

[5] Zob.: Ron Rivest, Robert Silverman, *Are 'Strong' Primes Needed for RSA?*, Cryptology ePrint Archive Report 2001/007.

[6] Losowałem ciąg małych liczb pierwszych, liczyłem ich iloczyn, dodawałem jedynkę i sprawdzałem, czy wyszła liczba pierwsza (formalnie pseudopierwsza). Postępowałem więc analogicznie do generowania odpornych liczb pierwszych, tylko odwrotnie. Jak zwykle przy generowaniu liczb pseudopierwszych o zadanych własnościach algorytm okazał się zaskakująco efektywny.