Applied Cryptography
CPEG 472/672
Lecture 10A

Instructor: Nektarios Tsoutsos

RSA Implementations

- Step 1:
  - Never implement RSA yourself
  - “20 Years of Attacks on the RSA Cryptosystem”
- Step 2:
  - Use a library for RSA
  - Requires an arbitrary precision arithmetic library (e.g., GMP)
  - Use secure padding (OAEP, PSS)

Modular Exponentiation

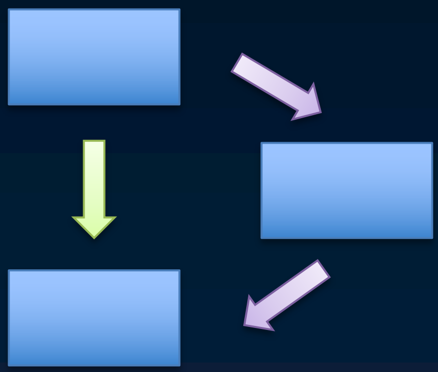
- $y = x^e \bmod n$
- Naïve method
  - Multiply x to itself e-1 times then mod n
  - Inefficient
- Square and multiply method
  - Exponentially faster than the naïve method
  - E.g., if $\mathrm{e} = 2^{16} + 1$ then $x^e \bmod n$ requires only 17 multiplications (not 65536)
  - Uses the bits of the exponent one by one

Square and Multiply

- Break the exponent e in bits
  - $e_{m-1}, e_{m-2}, e_{m-3}, \ldots, e_1, e_0$

```
expMod(x, e, n) {
 y = 1
 for i = m - 1 to 0 {
   y = y * y mod n
   if e_i == 1 then
      y = y * x mod n
 }
 return y
}
```

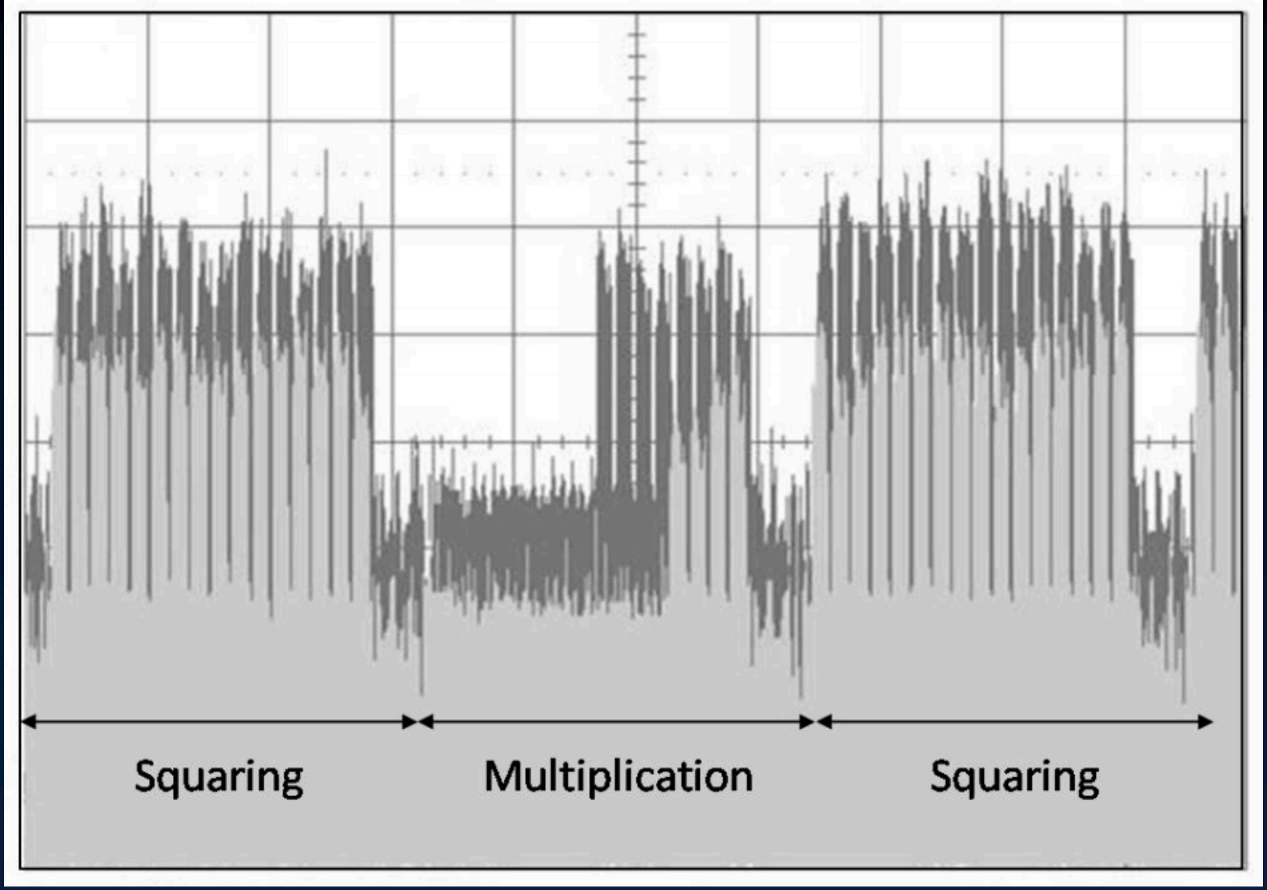
What is the problem?



Timing and power analysis attacks

- Exponentiation operations depend on the bits of exponent e
  - The algorithm uses a different path if e=1 compared to when e=0
  - When e=1 the loop is slower
  - Attackers can monitor execution time
- Power analysis attacks
  - Hardware-level attack
  - When e=1 the loop consumes more power

Power analysis attack



Benefit of small exponents e

- RSA encryption and signature verification uses exponentiation to e
  - Square and multiply cost depends on the bits of exponent e
  - Smaller e = faster exponentiation
  - Typically select e=65537
    - Only 17 multiplication needed
- In RSA, d is about the size of N
- Avoid using e=3
  - Low exponent attacks possible

Chinese Remainder Theorem (CRT)

- Assume $n = n_1 \cdot n_2 \cdot n_3 \ldots$
  - All factors are pairwise coprime
  - If we know $x \bmod n_1$, $x \bmod n_2$, $x \bmod n_3$, … we can recover $x \bmod n$
- $x \bmod n = \sum P(n_i)$
  - $P(n_i) = (x \bmod n_i) \cdot \frac{n}{n_i} \cdot \left(\left(\frac{n}{n_i}\right)^{-1} \bmod n_i \right) \bmod n$
  - CRT can accelerate RSA Dec and Signing by 4x
  - We can recover $x^d \bmod n$
    - $x_p = x^{d \bmod (p-1)} \bmod p$ and $x_q = x^{d \bmod (q-1)} \bmod q$
    - $x = x_p \cdot \boxed{q \cdot (q^{-1} \bmod p)} + x_q \cdot \boxed{p \cdot (p^{-1} \bmod q)} \bmod n$

Attacks

- Bellcore Fault Injection Attack (RSA-CRT)
  - RSA FDH signatures (deterministic)
  - Very powerful attack, recovers p (or q)
  - Attacker injects a fault in the CRT computation of x_q (or x_p)
    - Using faulty x_q' we recover faulty x'
    - Attacker can recover $p = \mathrm{GCD}(x - x', n),\ q = n/p$
      - $x - x' = \left(x_q - x_q'\right) \cdot p \cdot (p^{-1} \bmod q) \bmod n$, so $p | x - x'$
- Sharing your d and n
  - Possible to recover p and q using d, n

Factorize n using d

- $e \cdot d = 1 + k \cdot \varphi(n)$ for some k
- For a coprime to n, $a^{k\varphi(n)} = 1 \bmod n$
- $k\varphi(n) = k(p-1)(q-1)$ is even integer
  - $k\varphi(n) = 2^s t$ for some s and t
  - $\left(a^{t2^{s-1}}\right)^2 = 1 \bmod n$, which is an equation in the form $x^2 = 1$ that has solutions $x - 1, x + 1$
  - So, $a^{t2^{s-1}} - 1$ is a factor of n (either p or q)
    - We can run a program to find a, t and s
    - Demo today

Hands-on exercises

- Modular Exponentiation (naïve method)
- Shift and Multiply Exponentiation
- Montgomery Ladder exponentiation
- Recovering p, q from d, n

Reading for next lecture

- Aumasson: Chapter 11 until Anonymous Diffie-Hellman (inclusive)
  - We will have a short quiz on the material