Modular Exponentiation Computation & Vulnerability

Cybersecurity Specialization
-- Hardware Security

Computing a^e (mod n)

- Exponentiation and modular
  - $a^e = b$
  - b (mod n)
- Iterative exponentiation and modular
  - If $x \equiv y \pmod{n}$, then $ax \equiv ay \pmod{n}$
  - Modular whenever larger than n
- Question for hardware designer: can we multiply less than e-1 times?

Motivation

- How do we compute a^{16}?
  - $a^{16} = a*a*a*...*a$: multiply 15 times
  - $a^{16} = (a^8)^2 = (a^2)^2)^2)^2$: square 4 times
- How about a^{23}?
  - $a^{23} = (a^{16})(a^4)(a^2)(a^1)$: no need to do 22 multiplications
- Square and multiply algorithm
  - Left to right
  - Right to left

Square and Multiply Algorithm (1)

- Goal: Compute $a^e \pmod n$

1. convert e to binary: $k_s k_{s-1} ... k_1 k_0$
2. b = 1;
3. for (i=s; i>=o; i--)
4. { b = b*b (mod n);
5. if (k_i == 1)
6. b = b * a (mod n)
7. }
8. return b;

Square and Multiply Algorithm (2)

- Goal: Compute $a^e \pmod n$

1. convert e to binary: $k_s k_{s-1} \ldots k_1 k_0$
2. b = a^{k0}; c = a;
3. for (i=1; i<=s; i++)
4. { c = c*c (mod n);
5. if (k_i == 1)
6. b = b * c (mod n)
7. }
8. return b;

Square and Multiply Algorithm

- Goal: Compute $a^e \pmod n$

1. convert e to binary: $k_s k_{s-1} \ldots k_1 k_0$
2. b = 1;
3. for (i=s; i>=0; i--)
4. { b = b*b (mod n);
5. if (k_i == 1)
6. b = b * a (mod n)
7. }
8. return b;

Side channel attacks!

Observable side channel info during hardware execution: current, power, timing, ...

The value of bit k_i determines whether this non-trivial operation will be required.