

RSA Implementation Attacks

RSA

- RSA

- Public key: (e, N)

- Private key: d

- Encrypt M

$$C = M^e \pmod{N}$$

- Decrypt C

$$M = C^d \pmod{N}$$

- Digital signature

- Sign $h(M)$

- In protocols,
sign M itself:

$$S = M^d \pmod{N}$$

Implementation Attacks

- ❑ Attacks on RSA implementation
 - Not attacks on RSA algorithm per se
- ❑ Timing attacks
 - Exponentiation is very expensive computation
 - Try to exploit differences in timing related to differences in private key bits
- ❑ Glitching attack
 - Induced errors may reveal private key

Modular Exponentiation

- ❑ Attacks we discuss arise from precise details of modular exponentiation
- ❑ For efficiency, modular exponentiation uses some combination of
 - Repeated squaring
 - Sliding window
 - Chinese Remainder Theorem (CRT)
 - Montgomery multiplication
 - Karatsuba multiplication
- ❑ Next, we briefly discuss each of these

Repeated Squaring

- ❑ Modular exponentiation example
 - $5^{20} = 95367431640625 = 25 \pmod{35}$
- ❑ A better way: **repeated squaring**
 - $20 = 10100$ base 2
 - $(1, 10, 101, 1010, 10100) = (1, 2, 5, 10, 20)$
 - Note that $2 = 1 \cdot 2$, $5 = 2 \cdot 2 + 1$, $10 = 2 \cdot 5$, $20 = 2 \cdot 10$
 - $5^1 = 5 \pmod{35}$
 - $5^2 = (5^1)^2 = 5^2 = 25 \pmod{35}$
 - $5^5 = (5^2)^2 \cdot 5^1 = 25^2 \cdot 5 = 3125 = 10 \pmod{35}$
 - $5^{10} = (5^5)^2 = 10^2 = 100 = 30 \pmod{35}$
 - $5^{20} = (5^{10})^2 = 30^2 = 900 = 25 \pmod{35}$
- ❑ No huge numbers and it is **efficient**
 - In this example, 5 steps vs 20 for naïve method

Repeated Squaring

□ Repeated Squaring algorithm

// Compute $y = x^d \pmod{N}$

// where, in binary, $d = (d_0, d_1, d_2, \dots, d_n)$ with $d_0 = 1$

$s = x$

for $i = 1$ to n

$s = s^2 \pmod{N}$

 if $d_i == 1$ then

$s = s \cdot x \pmod{N}$

 end if

next i

return s

Sliding Window

- ❑ A simple time memory tradeoff for repeated squaring
- ❑ Instead of processing each bit...
- ❑ ...process block of n bits at once
 - Use pre-computed lookup tables
 - Typical value is $n = 5$

Chinese Remainder Theorem

- ❑ Chinese Remainder Theorem (CRT)
- ❑ We want to compute $C^d \pmod{N}$ where $N = pq$
- ❑ With CRT, we compute C^d modulo p and modulo q , then “glue” them together
- ❑ Two modular reductions of size $N^{1/2}$
 - As opposed to one reduction of size N
- ❑ CRT provides significant speedup

CRT Algorithm

- We know C , d , N , p and q
- Want to compute
 $C^d \pmod{N}$ where $N = pq$
- Pre-compute
 $d_p = d \pmod{p-1}$ and $d_q = d \pmod{q-1}$
- And determine a and b such that
 $a = 1 \pmod{p}$ and $a = 0 \pmod{q}$
 $b = 0 \pmod{p}$ and $b = 1 \pmod{q}$

CRT Algorithm

- We have d_p, d_q, a and b satisfying
$$d_p = d \pmod{p-1} \text{ and } d_q = d \pmod{q-1}$$
$$a = 1 \pmod{p} \text{ and } a = 0 \pmod{q}$$
$$b = 0 \pmod{p} \text{ and } b = 1 \pmod{q}$$
- Given C , want to find $C^d \pmod{N}$
- Compute: $C_p = C \pmod{p}$ and $C_q = C \pmod{q}$
- And: $x_p = C_p^{d_p} \pmod{p}$ and $x_q = C_q^{d_q} \pmod{q}$.
- Solution is: $C^d \pmod{N} = (ax_p + bx_q) \pmod{N}$.

CRT Example

- ❑ Suppose $N = 33$, $p = 11$, $q = 3$ and $d = 7$
 - Then $e = 3$, but not needed here
- ❑ Pre-compute
$$d_p = 7 \pmod{10} = 7 \text{ and } d_q = 7 \pmod{2} = 1$$
- ❑ Also, $a = 12$ and $b = 22$ satisfy conditions
- ❑ Suppose we are given $C = 5$
 - That is, we want to compute $C^d = 5^7 \pmod{33}$
 - Find $C_p = 5 \pmod{11} = 5$ and $C_q = 5 \pmod{3} = 2$
 - And $x_p = 5^7 = 3 \pmod{11}$, $x_q = 2^1 = 2 \pmod{3}$
- ❑ Easy to verify: $5^7 = 3 \cdot 12 + 22 \cdot 2 = 14 \pmod{33}$

CRT: The Bottom Line

- ❑ Looks like a lot of work
- ❑ But it is actually a big “win”
 - Provides a speedup by a factor of 4
- ❑ Any disadvantage?
 - Factors p and q of N must be known
 - Violates “trap door” property?
 - Used only for private key operations

Montgomery Multiplication

- ❑ Very clever method to reduce work in modular multiplication
 - And therefore in modular exponentiation
- ❑ Consider computing $ab \pmod{N}$
- ❑ Expensive part is modular reduction
- ❑ Naïve approach requires division
- ❑ In some cases, no division needed...

Montgomery Multiplication

- ❑ Consider product $ab = c \pmod{N}$
 - Where modulus is of form $N = m^k - 1$
- ❑ Then there exist c_0 and c_1 such that
$$c = c_1 m^k + c_0$$
- ❑ Can rewrite this as
$$c = c_1(m^k - 1) + (c_1 + c_0) = c_1 + c_0 \pmod{N}$$
- ❑ In this case, if we can find c_1 and c_0 , then no division is required in modular reduction

Montgomery Multiplication

- For example, consider $3089 \pmod{99}$

$$\begin{aligned} 3089 &= 30 \cdot 100 + 89 \\ &= 30(100 - 1) + (30 + 89) \\ &= 30 \cdot 99 + (30 + 89) \\ &= 119 \pmod{99} \end{aligned}$$

- Only one subtraction required to compute $3089 \pmod{99}$
- In this case, no division needed

Montgomery Multiplication

- ❑ Montgomery analogous to previous example
 - But Montgomery works for any modulus N
 - Big speedup for modular exponentiation
- ❑ Idea is to convert to “Montgomery form”, do multiplications, then convert back
 - Montgomery multiplication is highly efficient way to do multiplication and modular reduction
 - In spite of conversions to and from Montgomery form, this is a BIG win for exponentiation

Montgomery Form

- ❑ Consider $ab \pmod{N}$
- ❑ Choose $R = 2^k$ with $R > N$ and $\gcd(R, N) = 1$
- ❑ Also, find R' and N' so that $RR' - NN' = 1$
- ❑ Instead of a and b , we work with
$$a' = aR \pmod{N} \text{ and } b' = bR \pmod{N}$$
- ❑ The numbers a' and b' are said to be in **Montgomery form**

Montgomery Multiplication

- Given

$$a' = aR \pmod{N}, b' = bR \pmod{N} \text{ and } RR' - NN' = 1$$

- Compute

$$a'b' = (aR \pmod{N})(bR \pmod{N}) = abR^2$$

- Due to “mod N”, actually have $abR^2 \pmod{N}$

 - But it may not lie between 0 and $N - 1$

- That is, abR^2 denotes the product $a'b'$ without an additional mod N reduction

- Note that abR^2 need not be divisible by R

Montgomery Multiplication

- Given

$a' = aR \pmod{N}$, $b' = bR \pmod{N}$ and $RR' - NN' = 1$

- Then $a'b' = (aR \pmod{N})(bR \pmod{N}) = abR^2$

- Want $a'b'$ to be in Montgomery form

- That is, want $abR \pmod{N}$, not abR^2

- Note that $RR' = 1 \pmod{N}$

- Looks easy, since $abR^2R' = abR \pmod{N}$

- But, want to avoid costly mod N operation

- Montgomery algorithm provides clever solution

Montgomery Multiplication

- ❑ Given abR^2 , $RR' - NN' = 1$ and $R = 2^k$
- ❑ Want to find $abR \pmod{N}$
 - Without costly mod N operation (division)
- ❑ Note: “mod R ” and division by R are easy
 - Since R is a power of 2
- ❑ Let $X = abR^2$
- ❑ **Montgomery algorithm** on next slide

Montgomery Reduction

- Have $X = abR^2$, $RR' - NN' = 1$, $R = 2^k$

- Want to find $abR \pmod{N}$

- **Montgomery reduction**

$$m = (X \pmod{R}) \cdot N' \pmod{R}$$

$$x = (X + mN)/R \pmod{R}$$

if $x \geq N$ then

$$x = x - N \text{ // extra reduction}$$

end if

return x

Montgomery Reduction

- ❑ Why does Montgomery reduction work?
 - Recall that input is $X = abR^2$
 - Claim: output is $x = abR \pmod{N}$
- ❑ Must carefully examine main steps of **Montgomery reduction** algorithm:
$$m = (X \pmod{R}) \cdot N' \pmod{R}$$
$$x = (X + mN)/R \pmod{R}$$

Montgomery Reduction

- ❑ Given $X = abR^2$ and $RR' - NN' = 1$
 - Note that $N'N = -1 \pmod{R}$
- ❑ Consider $m = (X \pmod{R}) \cdot N' \pmod{R}$
 - In words: m is product of N' and remainder of X/R
- ❑ Therefore, $X + mN = X - (X \pmod{R})$
 - Implies $X + mN$ divisible by R
 - Since $R = 2^k$, division is simply a shift
- ❑ Consequently, it is trivial to compute
$$x = (X + mN)/R \pmod{R}$$

Montgomery Reduction

- Given $X = abR^2$ and $RR' - NN' = 1$
 - Note that $R'R = 1 \pmod{N}$
- Consider $x = (X + mN)/R \pmod{R}$
- Then $xR = X + mN = X \pmod{N}$
- And $xRR' = XR' \pmod{N}$
- Therefore
$$x = xRR' = XR' = abR^2R' = abR \pmod{N}$$

Montgomery Example

- ❑ Suppose $N = 79$, $a = 61$ and $b = 5$
- ❑ Use Montgomery to compute $ab \pmod{N}$
- ❑ Choose $R = 10^2 = 100$
 - For human readability, R is a power of 10
 - For computer, choose R to be a power of 2
- ❑ Then
$$a' = 61 \cdot 100 = 17 \pmod{79}$$
$$b' = 5 \cdot 100 = 26 \pmod{79}$$

Montgomery Example

- ❑ Consider $ab = 61 \cdot 5 \pmod{79}$
 - Recall that $R = 100$
 - So $a' = aR = 17 \pmod{79}$ and $b' = bR = 26 \pmod{79}$
- ❑ Euclidean Algorithm gives
$$64 \cdot 100 - 81 \cdot 79 = 1$$
- ❑ Then $R' = 64$ and $N' = 81$
- ❑ Monty reduction to determine $abR \pmod{79}$
- ❑ First, $X = a'b' = 17 \cdot 26 = 442 = abR^2$

Montgomery Example

- Given $X = a'b' = abR^2 = 442$
- Also have $R' = 64$ and $N' = 81$
- Want to determine $abR \pmod{79}$
- By Montgomery reduction algorithm
$$m = (X \pmod{R}) \cdot N' \pmod{R}$$
$$= 42 \cdot 81 = 3402 = 2 \pmod{100}$$
$$x = (X + mN)/R \pmod{R}$$
$$= (442 + 2 \cdot 79)/100 = 600/100 = 6 \pmod{100}$$
- Verify: $abR = 61 \cdot 5 \cdot 100 = 6 \pmod{79}$

Montgomery Example

- ❑ Have $abR = 6 \pmod{79}$
- ❑ But this number is in Montgomery form
- ❑ Convert to non-Montgomery form
 - Recall $R'R = 1 \pmod{N}$
 - So $abRR' = ab \pmod{N}$
- ❑ For this example, $R' = 64$ and $N = 79$
- ❑ Find $ab = abRR' = 6 \cdot 64 = 68 \pmod{79}$
- ❑ Easy to verify $ab = 61 \cdot 5 = 68 \pmod{79}$

Montgomery: Bottom Line

- ❑ Easier to compute $ab \pmod{N}$ directly, without using Montgomery algorithm!
- ❑ However, for exponentiation, Montgomery is much more efficient
 - For example, to compute $M^d \pmod{N}$
- ❑ To compute $M^d \pmod{N}$
 - Convert M to Montgomery form
 - Do repeated (cheap) Montgomery multiplications
 - Convert final result to non-Montgomery form

Karatsuba Multiplication

- ❑ Most efficient way to multiply two numbers of about same magnitude
 - Assuming "+" is much cheaper than "*"
- ❑ For n-bit number
 - Karatsuba work factor: $n^{1.585}$
 - Ordinary "long" multiplication: n^2
- ❑ Based on a simple observation...

Karatsuba Multiplication

- Consider the product

$$(a_0 + a_1 \cdot 10)(b_0 + b_1 \cdot 10)$$

- Naïve approach requires 4 multiplies to determine coefficients:

$$a_0b_0 + (a_1b_0 + a_0b_1)10 + a_1b_1 \cdot 10^2$$

- Same result with just 3 multiplies:

$$a_0b_0 + [(a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1]10 + a_1b_1 \cdot 10^2$$

Karatsuba Multiplication

- ❑ Does Karatsuba work for bigger numbers?

- ❑ For example

$$c_0 + c_1 \cdot 10 + c_2 \cdot 10^2 + c_3 \cdot 10^3 = C_0 + C_1 \cdot 10^2$$

- ❑ Where

$$C_0 = c_0 + c_1 \cdot 10 \text{ and } C_1 = c_2 + c_3 \cdot 10$$

- ❑ Can apply Karatsuba recursively to find product of numbers of any magnitude

Timing Attacks

- ❑ We discuss 3 different attacks
- ❑ Kocher's attack
 - Systems that use repeated squaring but not CRT or Montgomery (e.g., smart cards)
- ❑ Schindler's attack
 - Repeated squaring, CRT and Montgomery (no real systems use this combination)
- ❑ Brumley-Boneh attack
 - CRT, Montgomery, sliding windows, Karatsuba (e.g., openssl)

Kocher's Attack

- ❑ Attack on repeated squaring
 - Does not work if CRT or Montgomery used
 - In most applications, CRT and Montgomery multiplication are used
 - Some resource-constrained devices only use repeated squaring
- ❑ This attack aimed at smartcards

Repeated Squaring

□ Repeated Squaring algorithm

```
// Compute  $y = x^d \pmod{N}$   
// where, in binary,  $d = (d_0, d_1, d_2, \dots, d_n)$  with  $d_0 = 1$   
s = x  
for i = 1 to n  
    s =  $s^2 \pmod{N}$   
    if  $d_i == 1$  then  
        s =  $s \cdot x \pmod{N}$   
    end if  
next i  
return s
```

Kocher's Attack: Assumptions

- ❑ Repeated squaring algorithm is used
- ❑ Timing of multiplication $s \cdot x \pmod{N}$ in algorithm varies depending on s and x
 - That is, multiplication is not constant-time
- ❑ Trudy can accurately emulate timings given putative s and x
- ❑ Trudy can obtain accurate timings of private key operation, $C^d \pmod{N}$

Kocher's Attack

- ❑ Recover private key bits one (or a few) at a time
 - Private key: $d = d_0, d_1, \dots, d_n$ with $d_0 = 1$
 - Recover bits in order, d_1, d_2, d_3, \dots
- ❑ Do not need to recover all bits
 - Can efficiently recover low-order bits when enough high-order bits are known
 - Coppersmith's algorithm

Kocher's Attack

- Suppose bits d_0, d_1, \dots, d_{k-1} , are known
- We want to determine bit d_k
- Randomly select C_j for $j = 0, 1, \dots, m-1$, obtain timings $T(C_j)$ for $C_j^d \pmod{N}$
- For each C_j emulate steps $i = 1, 2, \dots, k-1$ of repeated squaring
- At step k , emulate $d_k = 0$ and $d_k = 1$
- **Variance** of timing difference will be smaller for correct choice of d_k

Kocher's Attack

- ❑ For example
 - Suppose private key is 8 bits
 - That is, $d = (d_0, d_1, \dots, d_7)$ with $d_0 = 1$
- ❑ Trudy is sure that $d_0 d_1 d_2 d_3 \in \{1010, 1001\}$
- ❑ Trudy generates random C_j , for each...
 - She obtains the timing $T(C_j)$ and
 - Emulates $d_0 d_1 d_2 d_3 = 1010$ and $d_0 d_1 d_2 d_3 = 1001$
- ❑ Let τ_i be emulated timing for bit i
 - Depends on bit value that is emulated

Kocher's Attack

- ❑ Private key is 8 bits
- ❑ Trudy is sure that $d_0d_1d_2d_3 \in \{1010, 1001\}$
- ❑ Trudy generates random C_j , for each...
- ❑ Define τ_i to be emulated timing for bit i
 - For $i < m$ let $\tau_{i...m}$ be shorthand for $\tau_i + \tau_{i+1} + \dots + \tau_m$
- ❑ Trudy tabulates $T(C_j)$ and $\tau_{0...3}$
- ❑ She computes variances
 - Smaller variance "wins"
- ❑ See next slide for fictitious example...

Kocher's Attack

- Suppose Trudy obtains timings

j	$T(C_j)$	emulate 1010		emulate 1001	
		$\tilde{t}_{0...3}$	$T(C_j) - \tilde{t}_{0...3}$	$\tilde{t}_{0...3}$	$T(C_j) - \tilde{t}_{0...3}$
0	12	5	7	7	5
1	11	5	6	4	7
2	12	6	6	7	5
3	13	8	5	6	7

- For $d_0d_1d_2d_3 = 1010$ Trudy finds

$$E(T(C_j) - \tau_{0...3}) = 6 \text{ and } \text{var}(T(C_j) - \tau_{0...3}) = 1/2$$

- For $d_0d_1d_2d_3 = 1001$ Trudy finds

$$E(T(C_j) - \tau_{0...3}) = 6 \text{ and } \text{var}(T(C_j) - \tau_{0...3}) = 1$$

- Kocher's attack implies $d_0d_1d_2d_3 = 1010$

Kocher's Attack

- ❑ Why does small variance win?
 - More bits are correct, so less variance
- ❑ More precisely, define
 - τ_i == emulated timing for bit i
 - t_i == actual timing for bit i
 - Assume $\text{var}(t_i) = \text{var}(t)$ for all i
 - u == measurement "error"
- ❑ In the previous example,
 - Correct case: $\text{var}(T(C_j) - \tau_{0\dots 3}) = 4\text{var}(t) + \text{var}(u)$
 - Incorrect case: $\text{var}(T(C_j) - \tau_{0\dots 3}) = 6\text{var}(t) + \text{var}(u)$

Kocher's Attack: Bottom Line

- ❑ Simple and elegant attack
 - Works provided only repeated squaring used
 - Limited utility—most RSA use CRT, Monty, etc.
- ❑ Why does this fail if CRT, etc., used?
- ❑ Timing variations due to CRT, Montgomery, etc., included in error term u
- ❑ Then $\text{var}(u)$ would overwhelm variance due to repeated squaring
 - We see precisely why this is so later...

Schindler's Attack

- ❑ Assume repeated squaring, Montgomery algorithm and CRT are all used
- ❑ Not aimed at any real system
 - Optimized systems also use Karatsuba for numbers of same magnitude and “long” multiplication for other numbers
 - Schindler's attack will not work in such cases
- ❑ But this attack is an important stepping stone to next attack (Brumley-Boneh)

Schindler's Attack

□ Montgomery algorithm

```
// Find Montgomery product  $a'b'$ 
// where  $a' = aR \pmod{N}$  and  $b' = bR \pmod{N}$ 
// Given  $RR' - NN' = 1$ 
Montgomery( $a', b'$ )
   $z = a'b'$ 
   $r = (z \pmod{R})N' \pmod{R}$ 
   $s = (z + rN)/R \pmod{N}$ 
  if  $s \geq N$  then
     $s = s - N$  // extra reduction
  end if
  return( $s$ )
end Montgomery
```

Schindler's Attack

□ Repeated squaring with Montgomery

```
// Find  $y = x^d \pmod{N}$ 
// where  $d = (d_0, d_1, d_2, \dots, d_{n-1})$  with  $d_0 = 1$ 
 $t' = xR \pmod{N}$  // Montgomery form
 $s' = t'$ 
for  $i = 1$  to  $n - 1$ 
     $s' = \text{Montgomery}(s', s')$ 
    if  $d_i == 1$  then
         $s' = \text{Montgomery}(s', t')$ 
    end if
next  $i$ 
 $t = s'R' \pmod{N}$  // convert to non-Montgomery form
return( $t$ )
```

Schindler's Attack

- CRT is also used
 - For each mod N reduction, where $N = pq$
 - Compute mod p and mod q reductions
 - Use repeated squaring algorithm on previous slide for both
- Trudy chooses ciphertexts C_j
 - Obtains accurate timings of $C_j^d \pmod{N}$
 - Goal is to recover d

Schindler's Attack

- ❑ Takes advantage of “extra reduction”
- ❑ Suppose $a' = aR \pmod{N}$ and B random
 - That is, B is uniform in $\{0, 1, 2, \dots, N-1\}$
- ❑ Schindler determined that

$$P(\text{extra reduction in Montgomery}(a', B)) = \frac{a'}{2R}.$$

$$P(\text{extra reduction in Montgomery}(B, B)) = \frac{N}{3R}.$$

Schindler's Attack

- Repeated squaring aka **square and multiply**
 - Square: $s' = \text{Montgomery}(s', s')$
 - Multiply: $s' = \text{Montgomery}(s', t')$

- Probability of extra reduction in “multiply”:

$$P(\text{extra reduction in Montgomery}(a', B)) = \frac{a'}{2R}.$$

- Probability of extra reduction in “square”:

$$P(\text{extra reduction in Montgomery}(B, B)) = \frac{N}{3R}.$$

Schindler's Attack

- Consider $a^d \pmod{N}$ using CRT
- First step is $a^{d_p} \pmod{p}$
- Where $d_p = d \pmod{(p-1)}$
- Suppose in this computation there are k_0 multiples and k_1 squares
- Expected number of extra reductions:

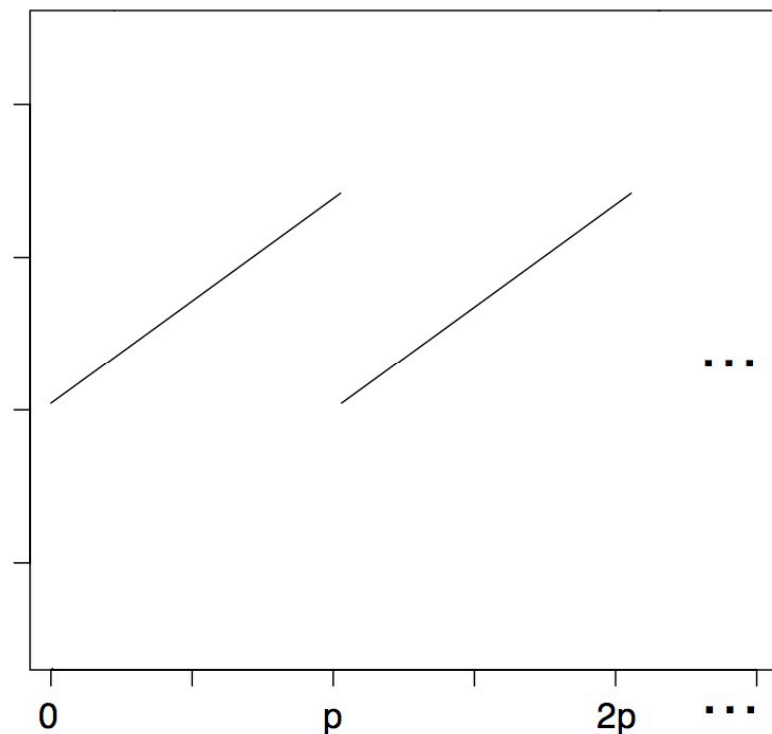
$$k_0 \frac{a' \pmod{p}}{2R} + k_1 \frac{p}{3R}.$$

Schindler's Attack

- Expected extra reductions:

$$k_0 \frac{a' \pmod{p}}{2R} + k_1 \frac{p}{3R}.$$

- Discontinuity at every integer multiple of p



Schindler's Attack

- ❑ How to take advantage of this?
- ❑ If chosen ciphertext C_0 is close to C_1
 - By continuity, timing $T(C_0)$ close to $T(C_1)$
- ❑ However, if $C_0 < kp < C_1$, then
$$|T(C_0) - T(C_1)|$$
is “large” due to discontinuity
- ❑ Note: total number of extra reductions include those for factors p and q
 - Discontinuities at all multiples of p and q

Schindler's Attack: Algorithm

- ❑ Select initial value x and offset Δ
- ❑ Let $C_i = x + i\Delta$ for $i = 0, 1, 2, \dots$
- ❑ Compute $t_i = T(C_{i+1}) - T(C_i)$ for $i = 0, 1, 2, \dots$
- ❑ Eventually, "bracket" a multiple of p
 - That is, $C_i < kp < C_{i+1}$
 - Detect this since t_i is large
- ❑ Then compute $\gcd(n, N)$ for all $C_i \leq n \leq C_{i+1}$
 - $\gcd(kp, N) = p$ and $\gcd(n, N) = 1$ otherwise

Schindler's: Bottom Line

- ❑ Clever attack if repeated squaring, Montgomery multiplication and CRT used
 - Crucial insight: extra reductions in Montgomery algorithm create timing issue
- ❑ However, attack not applicable to any real-world implementation
 - Optimized implementations also use Karatsuba
 - Karatsuba tends to counteract timing difference caused by extra reduction

Brumley-Boneh Attack

- ❑ CRT, Montgomery multiplication, sliding windows and Karatsuba
- ❑ Optimized RSA uses all of these
- ❑ Brumley-Boneh attack is robust
 - Works against OpenSSL over a network
 - Network timing variations are large
- ❑ The ultimate timing attack (to date)

Brumley-Boneh Attack

- ❑ Designed to attack RSA in OpenSSL
 - Highly optimized implementation
 - CRT, repeated squaring, Monty multiply, sliding window (5 bits)
 - Karatsuba multiply for numbers of same magnitude; long multiplication otherwise
- ❑ Kocher's attack fails due to CRT
- ❑ Schindler's attack fails due to Karatsuba
- ❑ Brumley-Boneh extends Schindler's attack

Brumley-Boneh Attack

- ❑ RSA in OpenSSL has two timing issues
 - Montgomery extra reductions
 - Karatsuba versus long multiplication
- ❑ These 2 tend to counteract each other
 - More extra reductions (slower) occur when Karatsuba multiply (faster) is used
 - Fewer extra reductions (faster) occur when long multiply (slower) is used

Brumley-Boneh Attack

- ❑ Consider C' , the Montgomery form of C
- ❑ Suppose C' is close to p with $C' > p$
 - Number of extra Montgomery reductions is small
 - Since $C' \pmod{p}$ is small, long multiply is used
- ❑ Suppose C' is close to p with $C' < p$
 - Number of extra Montgomery reductions is large
 - Since $C' \pmod{p}$ also close to p , Karatsuba multiply
- ❑ What to do?

Brumley-Boneh Attack

- ❑ Two timing effects: Montgomery extra reductions and Karatsuba effect
 - Each dominates at different points in attack
- ❑ Implies Schindler's could not recover bits where Karatsuba effect dominates
- ❑ Brumley-Boneh recovers factor p of modulus $N = pq$ one bit at a time
 - In this sense, analogous to Kocher's attack, but unlike Schindler's attack

Brumley-Boneh Attack: Step 1

- Denote bits of p as $p = (p_0, p_1, p_2, \dots, p_n)$
 - Where $p_0 = 1$
- Suppose p_1, p_2, \dots, p_{i-1} have been determined
- Choose $C_0 = (p_0, p_1, \dots, p_{i-1}, 0, 0, \dots, 0)$
- Choose $C_1 = (p_0, p_1, \dots, p_{i-1}, 1, 0, \dots, 0)$
- Note
 - If p_i is 1, then $C_0 < C_1 \leq p$
 - If p_i is 0, then $C_0 \leq p < C_1$

Brumley-Boneh Attack: Step 2

- ❑ Obtain decryption times $T(C_0)$ and $T(C_1)$
- ❑ Let $\Delta = |T(C_0) - T(C_1)|$
- ❑ If $C_0 < p < C_1$ then Δ is large $\Rightarrow p_i = 0$
- ❑ If $C_0 < C_1 < p$ then Δ is small $\Rightarrow p_i = 1$
 - Previous Δ used to set large/small thresholds
- ❑ Works provided that extra reduction or Karatsuba dominates at each step
 - See next slide...

Brumley-Boneh Attack: Step 2

- If $p_i = 1$ then $C_0 < C_1 < p$
 - Extra reductions are about the same
 - Karatsuba multiply used since mod p magnitudes are same
 - Expect Δ to be "small"
- If $p_i = 0$ then $C_0 < p < C_1$
 - If extra reduction dominate, $T(C_0) - T(C_1) > 0$
 - If Karatsuba vs long dominates, $T(C_0) - T(C_1) < 0$
 - In either case, expect Δ to be "large"

Brumley-Boneh Attack: Step 3

- ❑ Repeat steps 1 and 2
- ❑ Recover bits $p_{i+1}, p_{i+2}, p_{i+3}, \dots$
- ❑ When half of bits of p recovered, use Coppersmiths algorithm to factor N
- ❑ Then exponent d easily recovered

Brumley-Boneh Attack: Real-World Issues

- ❑ In OpenSSL, sliding windows used
 - Greatly reduces number of multiplies
 - Statistical methods must be used—repeated measurements, test nearby values, etc.
- ❑ OpenSSL attack over a network
 - Statistical methods needed
 - Attack is surprisingly robust
- ❑ Over realistic network, 1024-bit modulus factored with 1.4M chosen ciphertexts

Brumley-Boneh: Bottom Line

- ❑ A major cryptanalytic achievement
- ❑ Surprising that it is robust enough to overcome network variations
- ❑ Resulted in changes to OpenSSL
 - And other RSA implementations
- ❑ Brumley-Boneh is a realistic threat!

Preventing Timing Attack

- ❑ Several methods have been suggested
- ❑ Best solution is **RSA Blinding**
- ❑ To decrypt C generate random r then
$$Y = r^e C \pmod{N}$$
- ❑ Decrypt Y then multiply by $r^{-1} \pmod{N}$:
$$r^{-1} Y^d = r^{-1} (r^e C)^d = r^{-1} r C^d = C^d \pmod{N}$$
- ❑ Since r is random, Trudy cannot obtain timing info from choice of C
 - Slight performance penalty

Glitching Attack

- ❑ Induced error reveals private key
- ❑ CRT leads to simple glitching attack
- ❑ A single glitch may allow Trudy to factor the modulus!
- ❑ A realistic threat to smartcards
 - And other systems where attacker has physical access (e.g., trusted computing)

CRT

- Consider CRT for signing M
- Let $M_p = M \pmod{p}$ and $M_q = M \pmod{q}$
- Let $x_p = M_p^{d_p} \pmod{p}$ and $x_q = M_q^{d_q} \pmod{q}$
 $d_p = d \pmod{p-1}$ and $d_q = d \pmod{q-1}$
- Sign: $S = M^d \pmod{N} = ax_p + bx_q \pmod{N}$
 $a = 1 \pmod{p}$ and $a = 0 \pmod{q}$
 $b = 0 \pmod{p}$ and $b = 1 \pmod{q}$

Glitching Attack

- ❑ Trudy forces a single error to occur
- ❑ Suppose x'_q computed in place of x_q
 - But x_p computed correctly
 - That is, error in M_q or x_q computation
- ❑ "Signature" is $S' = ax_p + bx'_q \pmod{N}$
- ❑ Trudy knows error has occurred since $(S')^e \pmod{N} \neq M$

Glitching Attack

- ❑ Trudy has forced an error
- ❑ Trudy has $S' = ax_p + bx'_q \pmod{N}$
 - $a = 1 \pmod{p}$ and $a = 0 \pmod{q}$
 - $b = 0 \pmod{p}$ and $b = 1 \pmod{q}$
- ❑ Then $S' \pmod{p} = x_p = (M \pmod{p})^d \pmod{p-1}$
 - Follows from definitions of x_p and a

Glitching Attack

- ❑ Trudy has forced an error, so that
$$S' \pmod p = x_p = (M \pmod p)^d \pmod{(p-1)}$$
- ❑ It can be shown $(S')^e = M \pmod p$
 - That is, $(S')^e - M = kp$ for some k
- ❑ Also, $(S')^e \not\equiv M \pmod q$
 - Then $(S')^e - M$ **not** a multiple of the factor q
- ❑ Therefore, $\gcd(N, (S')^e - M)$ reveals nontrivial factor of N , namely, p

Glitching: Bottom Line

- ❑ Single glitch can break some systems
- ❑ A realistic threat
- ❑ Even if probability of error is small, advantage lies with attacker
- ❑ Glitches can also break some RSA implementations where CRT not used

Conclusions

- ❑ Timing attacks are real!
 - Serious issue for public key (symmetric key?)
- ❑ Glitching attacks also serious in some cases
- ❑ These attacks not traditional cryptanalysis
 - Here, Trudy does not play by the rules
- ❑ Crypto security—more than strong algorithms
 - Also need “strong” implementations
 - Good guys must think outside the box
 - Attackers will exploit any weak link