Remote Timing Attacks are Practical

by David Brumley and Dan Boneh

Presented by
Seny Kamara
in
Advanced Topics in Network Security (600/650.624)

Outline

- Traditional threat model in cryptography
- Side-channel attacks
- Kocher's timing attack
- Boneh & Brumley timing attack
- Experiments
- Countermeasures

Traditional Crypto

- Brute force attacks
  - large key
- Mathematical attacks
  - reduction to hard problem
  - RSAP: $(m^e \bmod n) \rightarrow m$
  - DHP: $(g^x, g^y) \rightarrow g^{xy}$

Traditional Crypto

- Attacker has access to:
  - Ciphertext
  - Algorithm

Real-Life Crypto

- Attacker has access to:
  - Ciphertext
  - Algorithm
  - Physical observables from the device

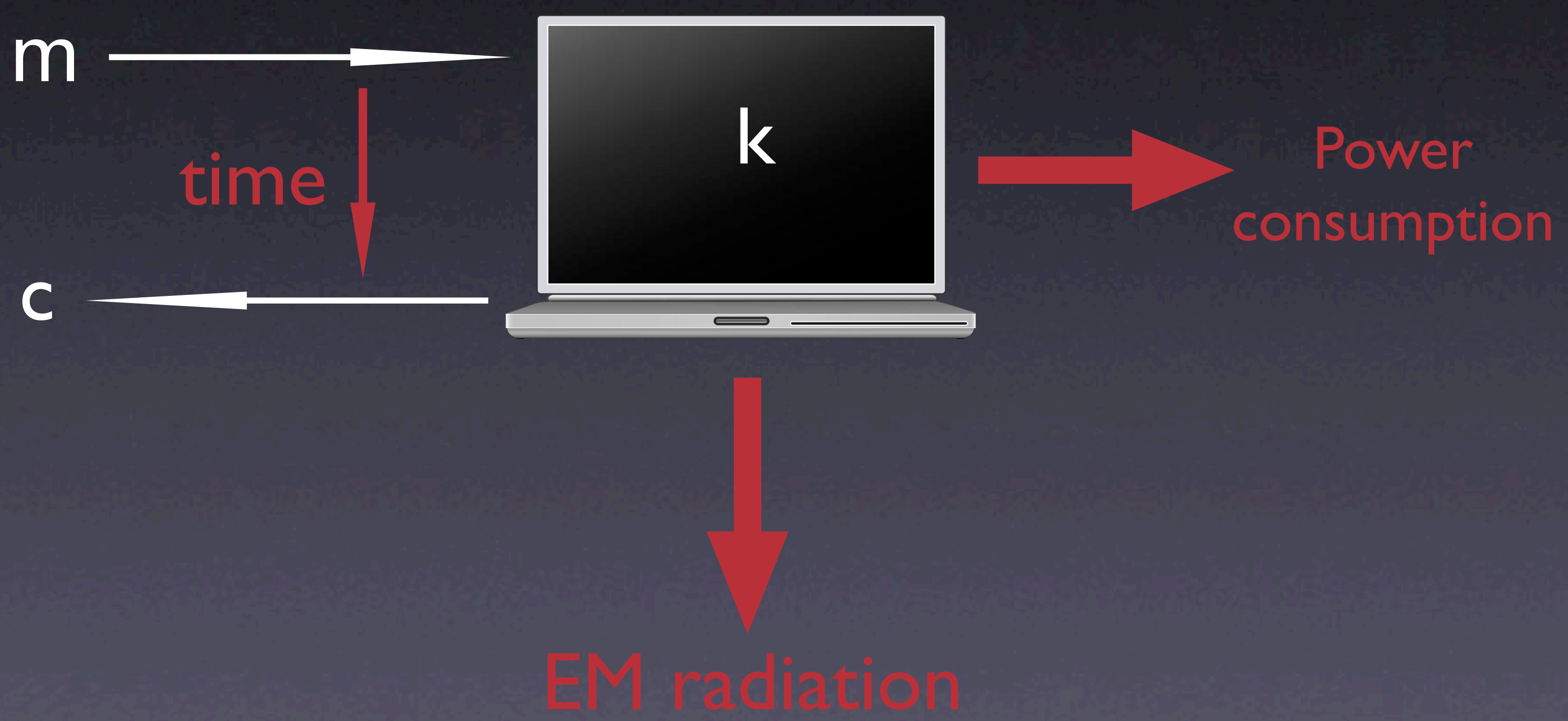
Side Channel Attacks

- Paul Kocher in 1996
- Recovers RSA and DSS signing key
- Not taken seriously by cryptographers
- Lot of attention from the press

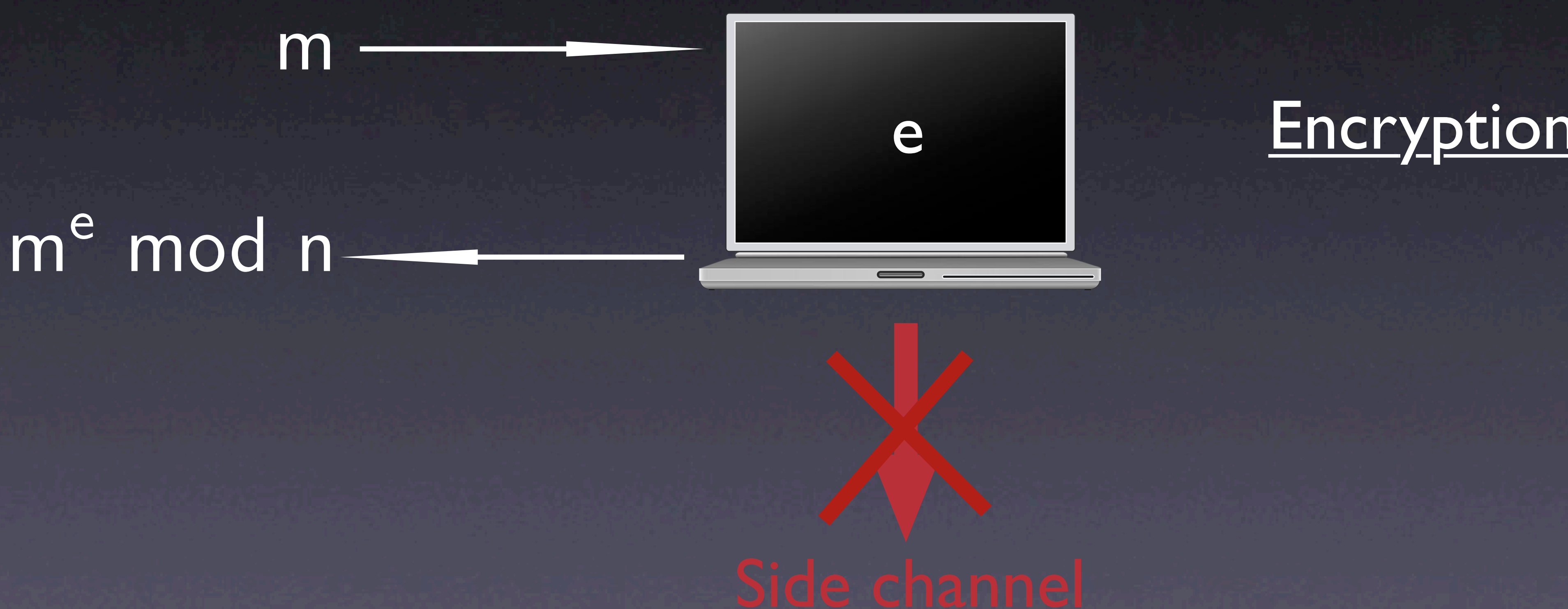
Side Channel Attacks

- Timing analysis
- Fault analysis
- Differential fault analysis
- Simple power analysis
- Differential power analysis
- EM analysis

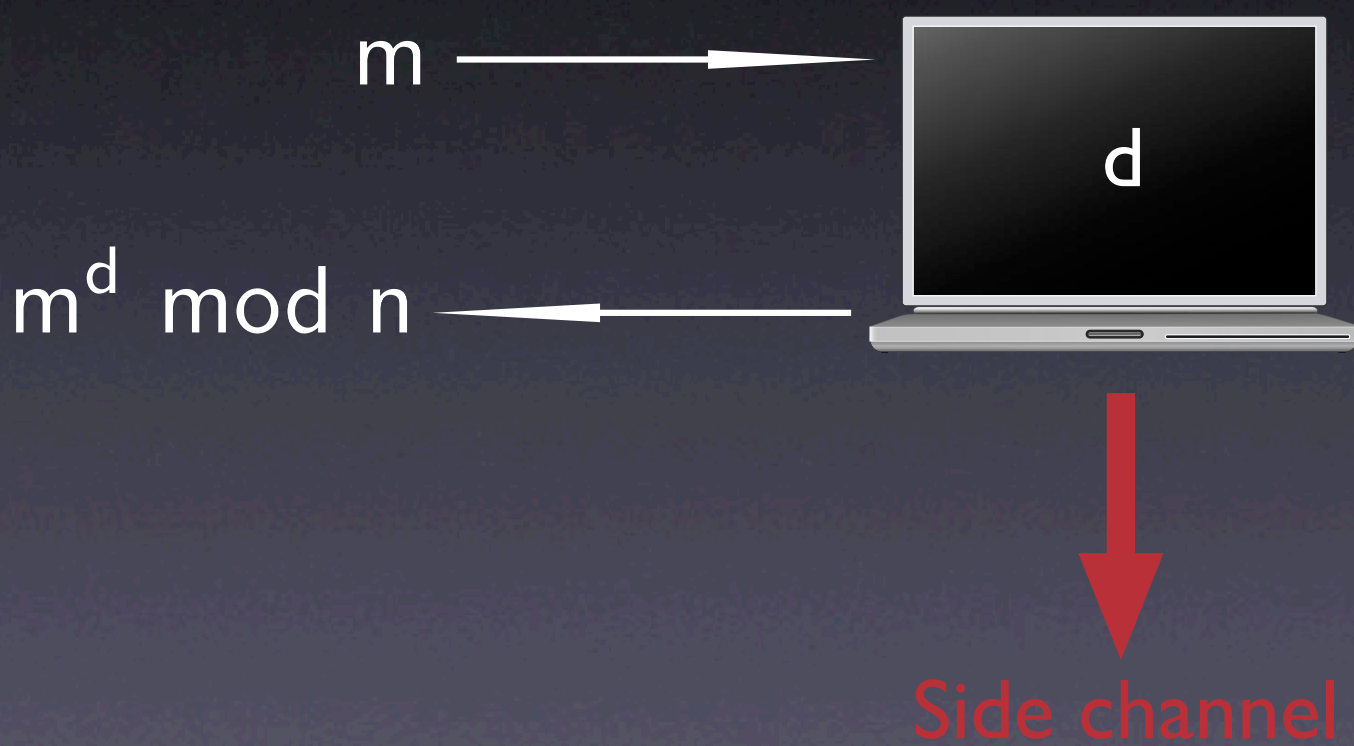
Side Channel Attacks



Side Channel Attacks



Side Channel Attacks



Decryption/
Signing

Kocher Timing Attack

- RSA signatures: $\mathsf{sig}(\mathsf{m}) = \mathsf{m}^{\mathsf{d}} \bmod \mathsf{n}$
- Modular exponentiation is computed using *square and multiply* algorithm
- Time of modular exponentiation is a function of the bits of the exponent
- Use time to recover exponent (signing key)

Kocher Timing Attack

- Recovers key bit by bit
- Guesses key bit then verifies
- Uses statistical analysis
- Needs many samples of signing time

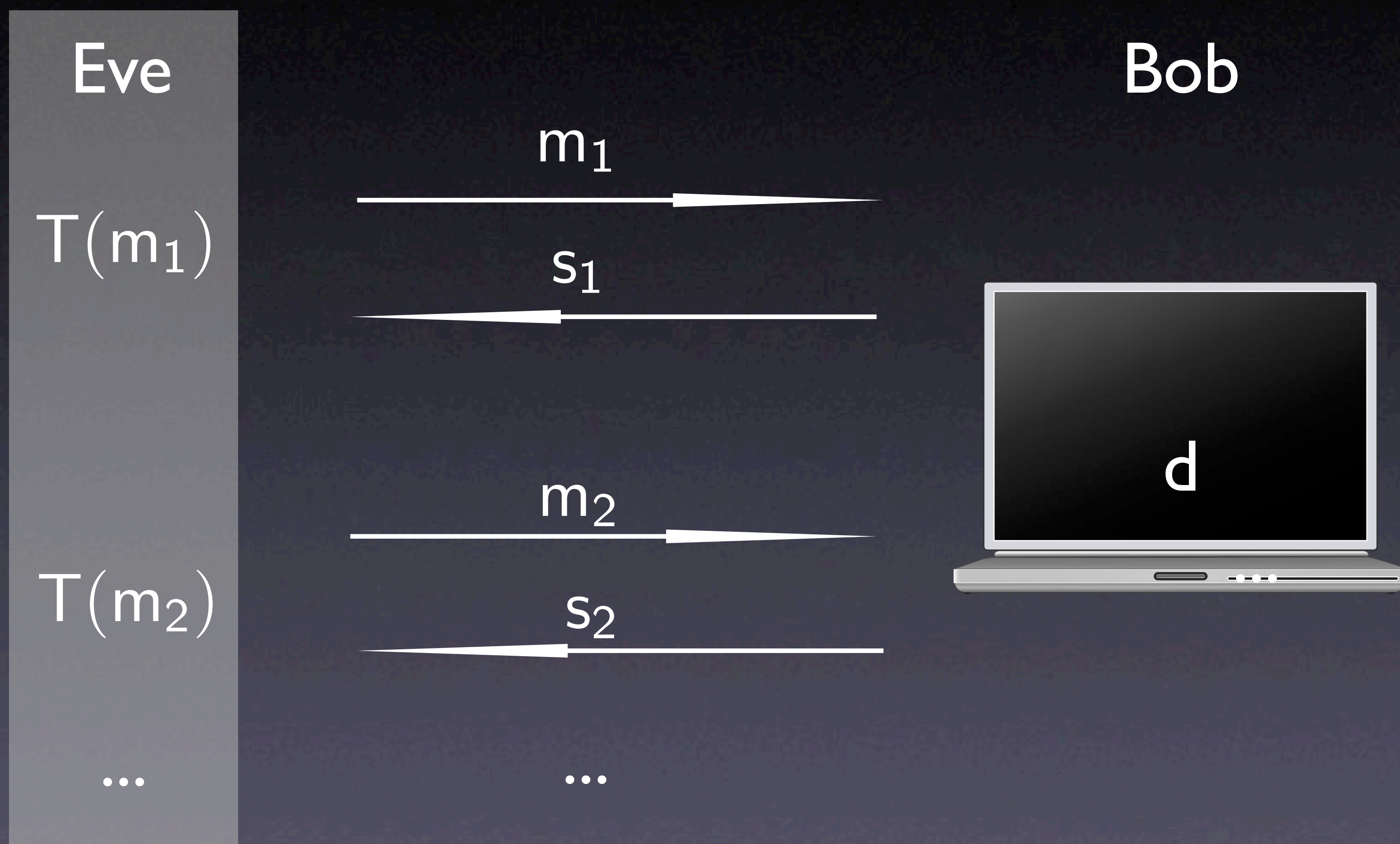
Kocher Attack Target

$$sig(m) = m^{d} \bmod n$$

Square and Multiply

```
INPUT: m, n, d
OUTPUT: x = m^d mod n
x := m
for i = n − 1 downto 0 do
    x := x^2
    if d_i = 1 then
        x := x · m mod n
    end if
end for
return x
```

Kocher Timing Attack



Kocher Timing Attack

Eve

$T^0(m_1)$

$T^0(m_2)$

...

m_1

s_1

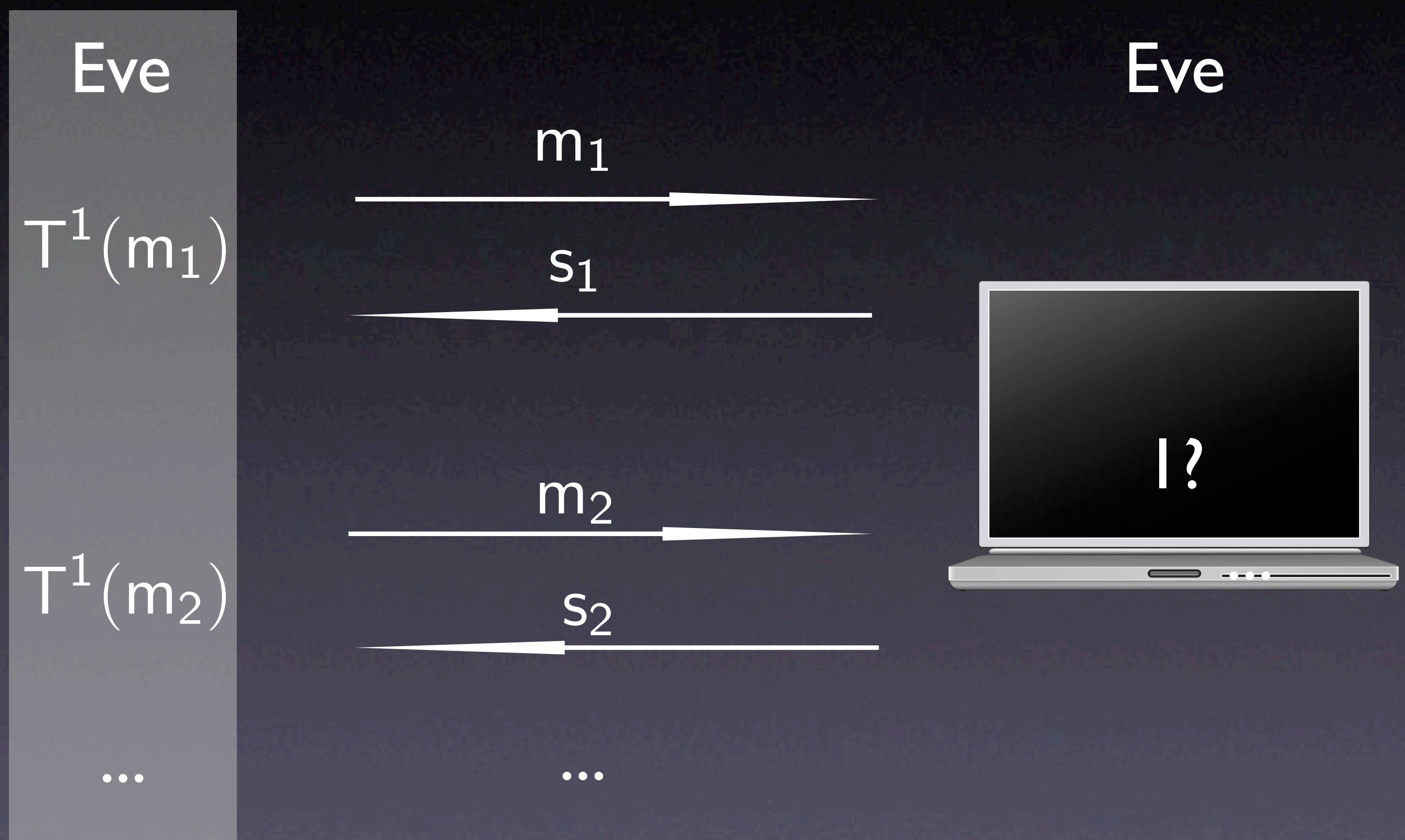
m_2

s_2

...

Eve

Kocher Timing Attack



Kocher Timing Attack

- Compare
  - $T(m_i)$ vs $T^0(m_i)$
  - $T(m_i)$ vs $T^1(m_i)$
- $T(m_i)$ will be correlated with correct guess

Kocher Timing Attack

- 1998 UCL experimental results:

Key size	sample size
64	1 500-6 500
128	12 000-20 000
256	70 000-80 000
512	350 000

Limit of Kocher Attack

- Does not work when mod exp is optimized

RSA with Sun Ze Th.

- $\text{sig}(m) = m^d \bmod n$
- Sun Ze Th. aka CRT
- m, d and n are order of 1024 bits
- exponentiation of 1024 bit number by another 1024 bit number taken modulo a third 1024 bit number

RSA with Sun Ze Th.

- exponentiate mod q (512 bits)
- exponentiate mod p (512 bits)
- combine using SZT to get mod n (= pq)

RSA with Sun Ze Th.

- $\text{sig}(m) = m^d \bmod n$ where $n = pq$
- $m_1 = m \bmod p$
- $m_2 = m \bmod q$
- $d_1 = d \bmod (p-1)$

  $d_2 = d \bmod (q-1)$

RSA with Sun Ze Th.

- $s_1 = m_1^{d_1} \mod p$
- $s_2 = m_2^{d_2} \mod q$
- $\mathrm{CRT}(s_1, s_2) = m^d \mod n$

RSA with Sun Ze Th.

- Modular exponentiation:
  - pre-processing
  - exponentiation mod p
  - exponentiation mod q
  - CRT

RSA with Sun Ze Th.

- Kocher's attack does not work
- Cannot get precise timings
- Cannot repeat pre-processing without factors
- Most implementations use CRT
- OpenSSL

OpenSSL

- SSL establishes encrypted and authenticated channel between client and server
- 1994
  - SSL v1 completed but never released
  - SSL v2 released with Navigator 1.1
  - SSL v2 PRNG broken

OpenSSL

- 1995
  - SSL v3 released (designed by Kocher)
  - SSL is ubiquitous
- 1996
  - IETF standardizes SSL

OpenSSL

- 1998
  - OpenSSL 0.9.1c is released (based on SSLeay)
  - mod_ssl for Apache is released

OpenSSL

- Most popular open source SSL implementation
- Most popular crypto library
- 18% of all Apache servers use mod_ssl
- stunnel
- sNFS

RSA in OpenSSL

- $\mathsf{sig(m) = m^d \bmod n}$
- *Sun Ze Theorem*
- Modular exponentiation: *sliding window*
- Modular reduction: *Montgomery*
- Multi-precision multiplication: *Karatsuba*

Sliding Window

- Extension of square and multiply
- uses multiple bits of the exponent at once
- makes attack more difficult

Montgomery Reduction

- Introduced in 1985 by Peter Montgomery
- Performs modular multiplication efficiently
- Transforms multiplication mod n to multiplication mod R

Montgomery Reduction

Algorithm 1 Montgomery Reduction

INPUT: x, y and q
OUTPUT: $x \cdot y \ mod \ q$
$RR^{-1} - qq^* = 1$
$\Psi(x) := xR \ mod \ q$
$\Psi(y) := yR \ mod \ q$
$z := \Psi(x) \times \Psi(y) = abR^2 \ mod \ q$
$r := z \times q^* \ mod \ R$
$s := \frac{z+rq}{R}$
if $s > q$ **then**
    $s := s - q$
end if
return s

extra reduction

Montgomery Reduction

- $\Pr[\text{extra reduction}] = \dfrac{m \bmod q}{2R}$

- $m = q \Rightarrow \Pr[\text{reduction}] = 0$
- $m \to q \Rightarrow \Pr[\text{reduction}] \nearrow$

  $m \to q+ \Rightarrow \Pr[\text{reduction}] \searrow$

Karatsuba

- Multi-precision multiplication
- $x \cdot y$ where $|x| = n$ and $|y| = n$
- Runs in $O(n^{\log_2 3})$
- As opposed to $O(n \cdot m)$
- worst case $O(n^2)$

Karatsuba

- Used *only* if inputs have same length
- OpenSSL:
  - if |x| = |y| then Karatsuba $O(n^{\log_2 3})$
  - if |x| != |y| then normal $O(n^2)$

Biases

- What is the effect of these optimizations on the exponentiation time?

Montgomery Reduction

- if m approaches q from below then slow
- if m approaches q from above then fast

Montgomery Reduction

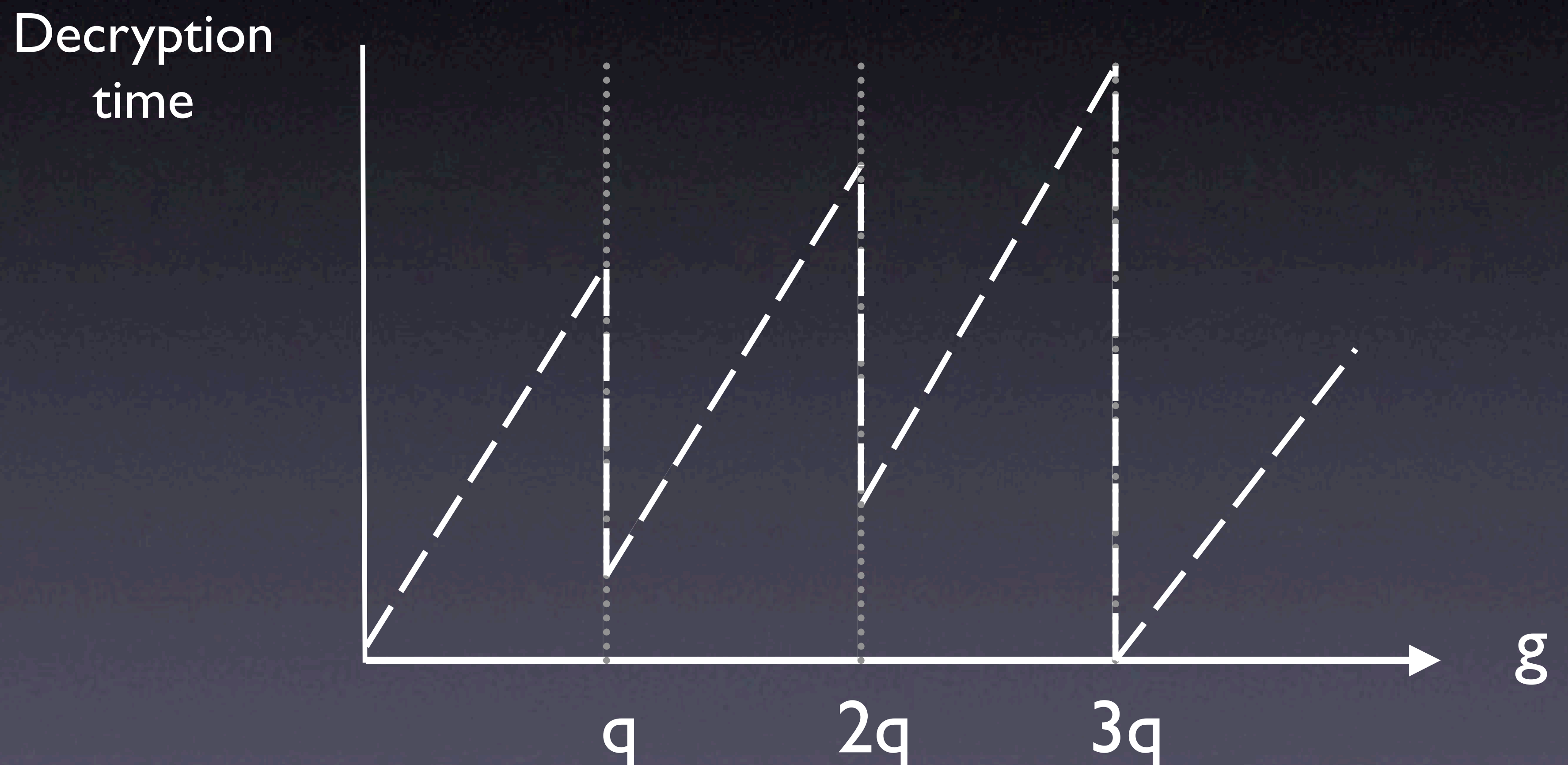
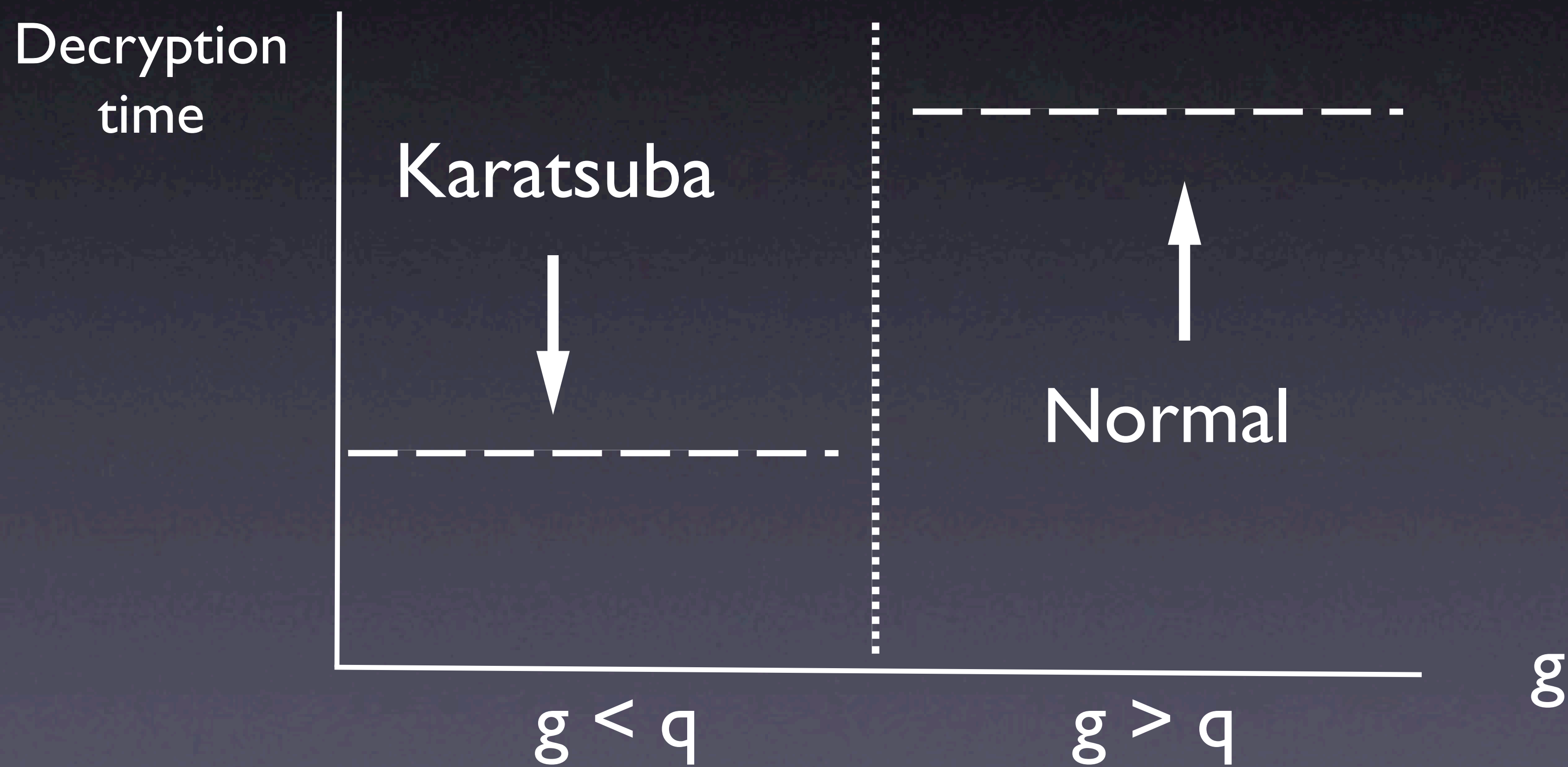


Figure 1

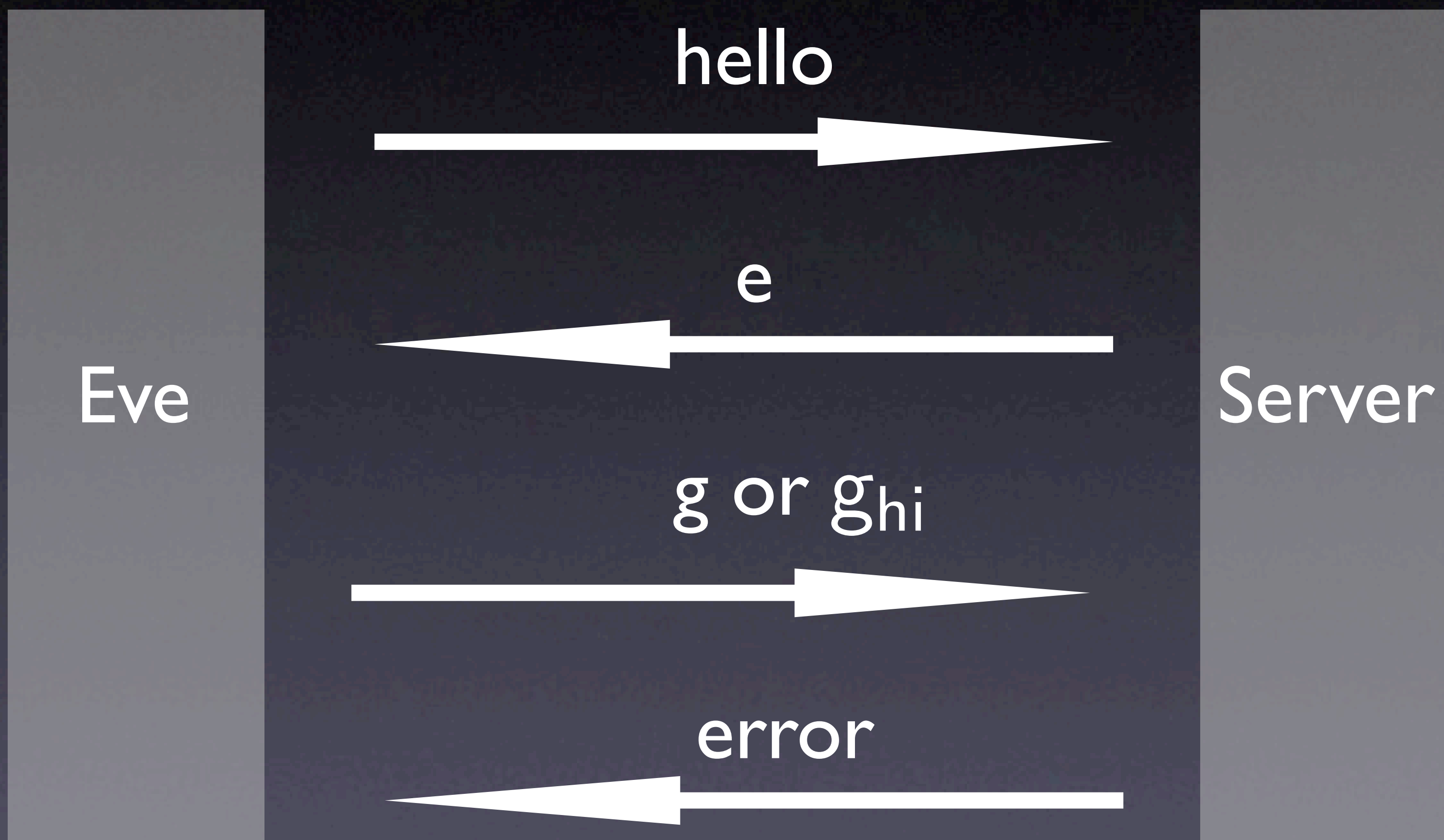
Multiplication

- if |x| = |y| then fast
- if |x| != |y| then slow

Multiplication



Boneh-Brumley Attack



Boneh-Brumley Attack

- Kocher attack recovers signing key
- Boneh-Brumley attack recovers factor

Kocher Attack Target

$$sig(m) = m^{d} \bmod n$$

Boneh-Brumley Target

$$\text{sig}(m) = m^d \bmod p \cdot q$$

Boneh-Brumley Target

- n = pq
- Knowing q we recover p

$$d = e^{-1} \bmod (p-1)(q-1)$$

Boneh-Brumley Attack

CRT → m modq

Square and multiply → $m^d \bmod q$

Montgomery → $m^d \bmod R$

Multiplication → $1 \cdot m$

Boneh-Brumley Attack

- $\mathsf{sig}(m) = m^d \bmod pq$
- Recover i^{th} bit of q
- when we already have the top $i-1$ bits

Timing Attack

- q: smallest factor
- g: same top $i-1$ bits as q (rest is all 0)
- g_{hi} : g with i^{th} bit set to 1
- Δ : decryption(g) - decryption(g_{hi})

Timing Attack

- i = 4
- q = 101 ?
- g = 101 0...
- g_{hi} = 101 10...

Timing Attack

- i = 4
- q = 101 1 ?
- g = 101 0...
- g_{hi} = 101 10...

if $q_4 = 1$ then $g < g_{hi} < q$

Timing Attack

- i = 4
- q = 101 0 ?
- g = 101 0...
- g_{hi} = 101 10...

$$\text{if } q_4 = 0 \text{ then } g < q < g_{hi}$$

Boneh-Brumley Attack

$$q_i = 0 \rightarrow g < q < g_{hi}$$

	Montgomery	Multiplication
$T(g)$	slow (xtra reds)	fast (kara)
$T(g_{hi})$	fast	slow (normal)
$\lvert\Delta\rvert$	large	large

Boneh-Brumley Attack

$$g < q < g_{hi}$$

	Montgomery	Multiplication
$T(g)$	slow (xtra reds)	fast (kara)
$T(g_{hi})$	fast	slow (normal)
$\|\Delta\|$	large	large

Boneh-Brumley Attack

$$q_i = 1 \rightarrow g < g_{hi} < q$$

	Montgomery	Multiplication
$T(g)$	slow	fast
$T(g_{hi})$	slow	fast
$\|\Delta\|$	small	small

Boneh-Brumley Attack

$$g < g_{hi} < q$$

	Montgomery	Multiplication
$T(g)$	slow	fast
$T(g_{hi})$	slow	fast
$\|\Delta\|$	small	small

Timing Attack

- if $q_4 = 1$ then $g < g_{hi} < q$ and
  - $|\Delta|$ is small
- if $q_4 = 0$ then $g < q < g_{hi}$ and
  - $|\Delta|$ is large

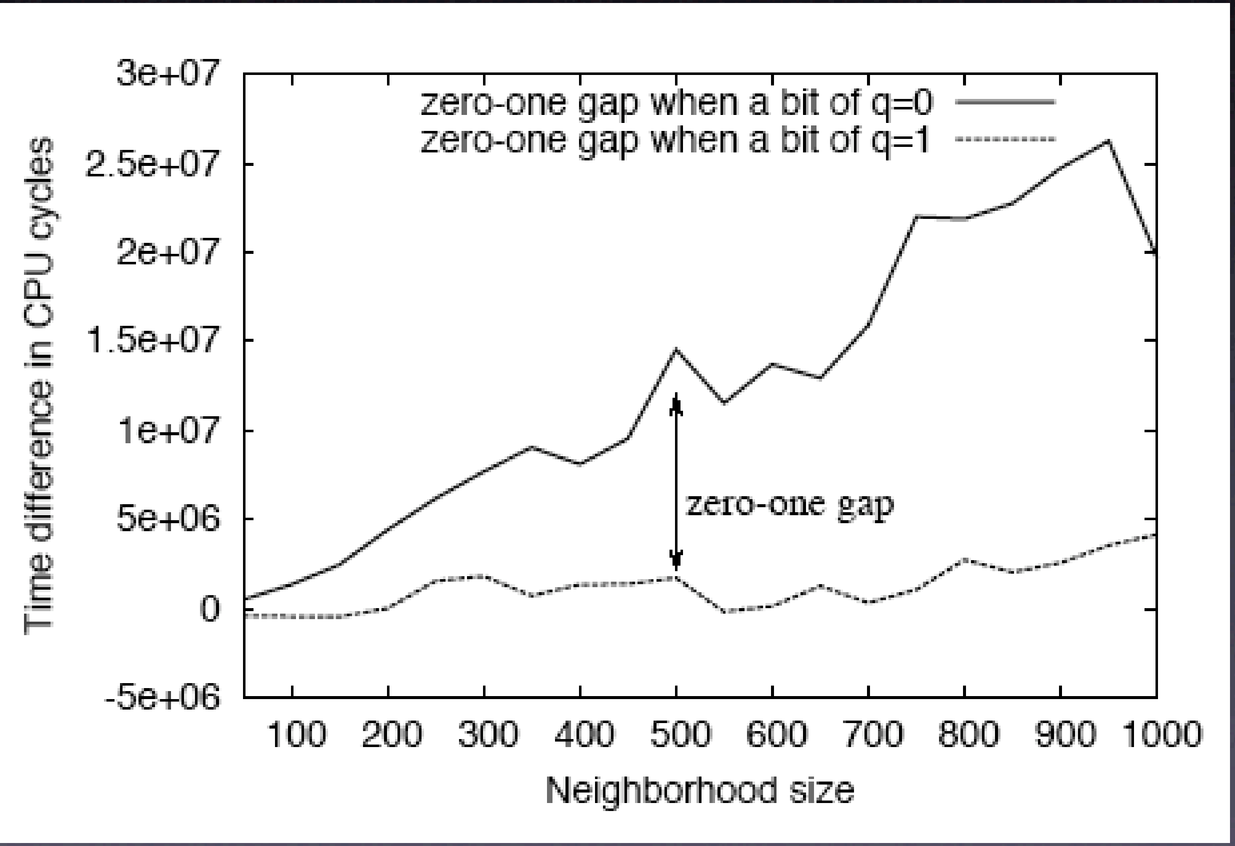
Experimental Setup

- RedHat Linux 7.3
- 2.4 GHz Pentium 4
- 1 GB of RAM
- gcc 2.96
- OpenSSL 0.9.7

Number of Queries

- Interprocess using TCP
- Neighborhood size: for each bit measure decryption time of many guesses (sliding window)
- Sample size: for each guess measure multiple times

Number of Queries



Number of Queries

- Delta increases as neighborhood size increases
- Variance decreases as sample size increases

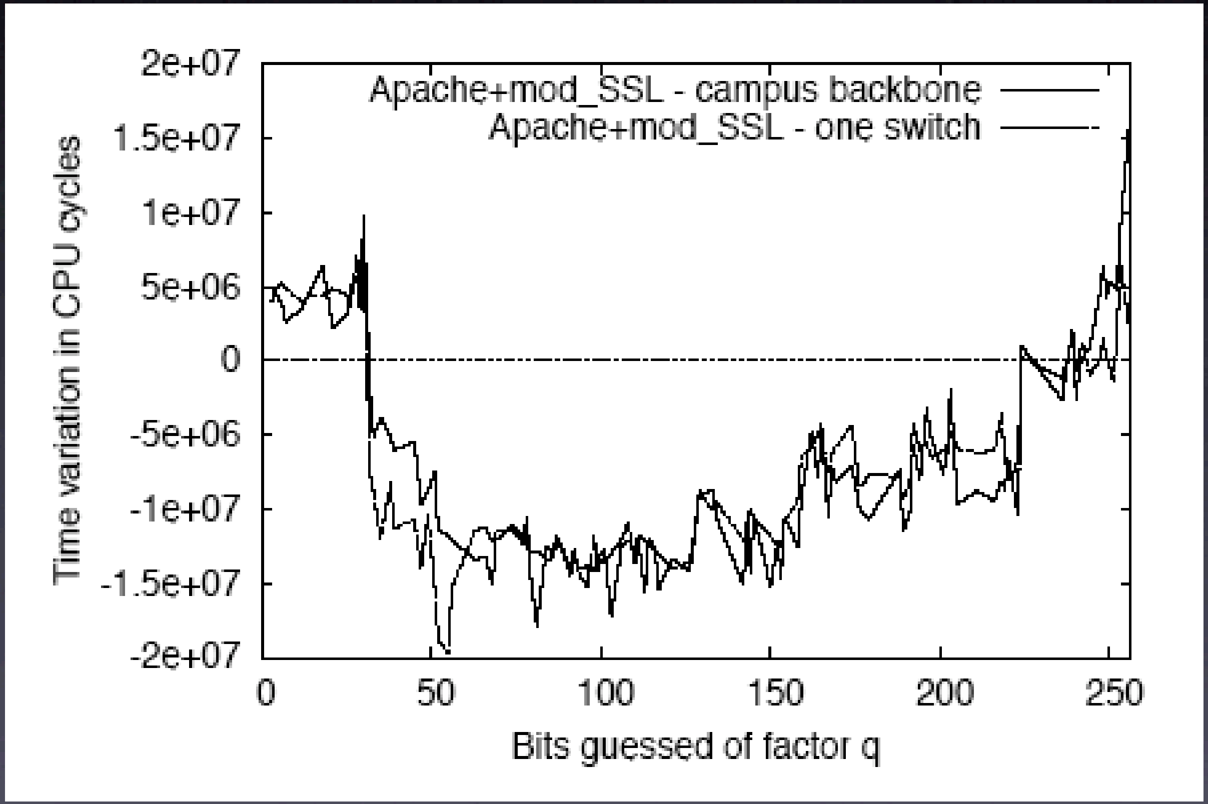
Other Experiments

- Tested using 3 different keys
- Deltas are very sensitive to
  - execution environment (cache misses, code offsets etc...)
  - compilation flags

Network Experiments

- Works against Apache+mod_ssl when seperated by:
  - 1 switch
  - 3 routers and a number of switches

Network


Apache+mod_SSL - campus backbone
Apache+mod_SSL - one switch
Time variation in CPU cycles
2e+07
1.5e+07
1e+07
5e+06
0
-5e+06
-1e+07
-1.5e+07
-2e+07
0
50
100
150
200
250
Bits guessed of factor q

Attack Results

- Interprocess attack
- 1024 bit key
- Unoptimized: 350 000 queries
- Optimized: 1.4 million queries
  - 2 hours

More Details

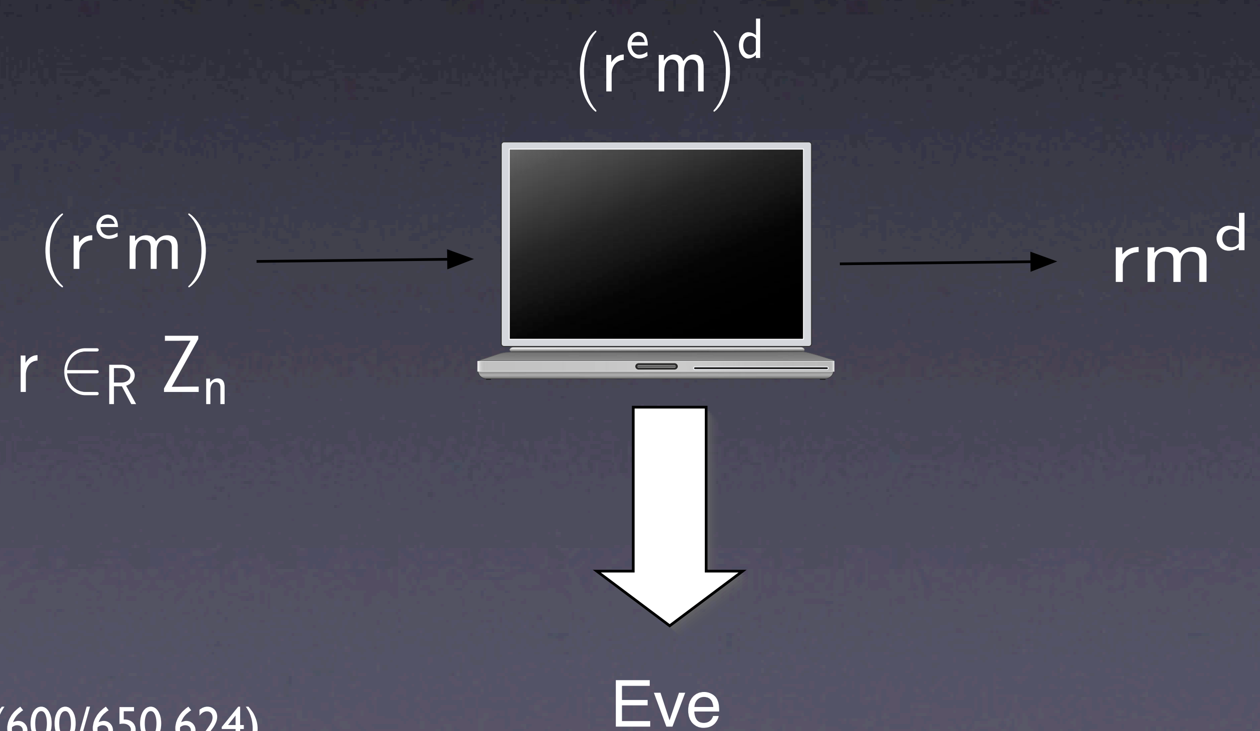
- Lucas will talk more about the experiments

Countermeasures

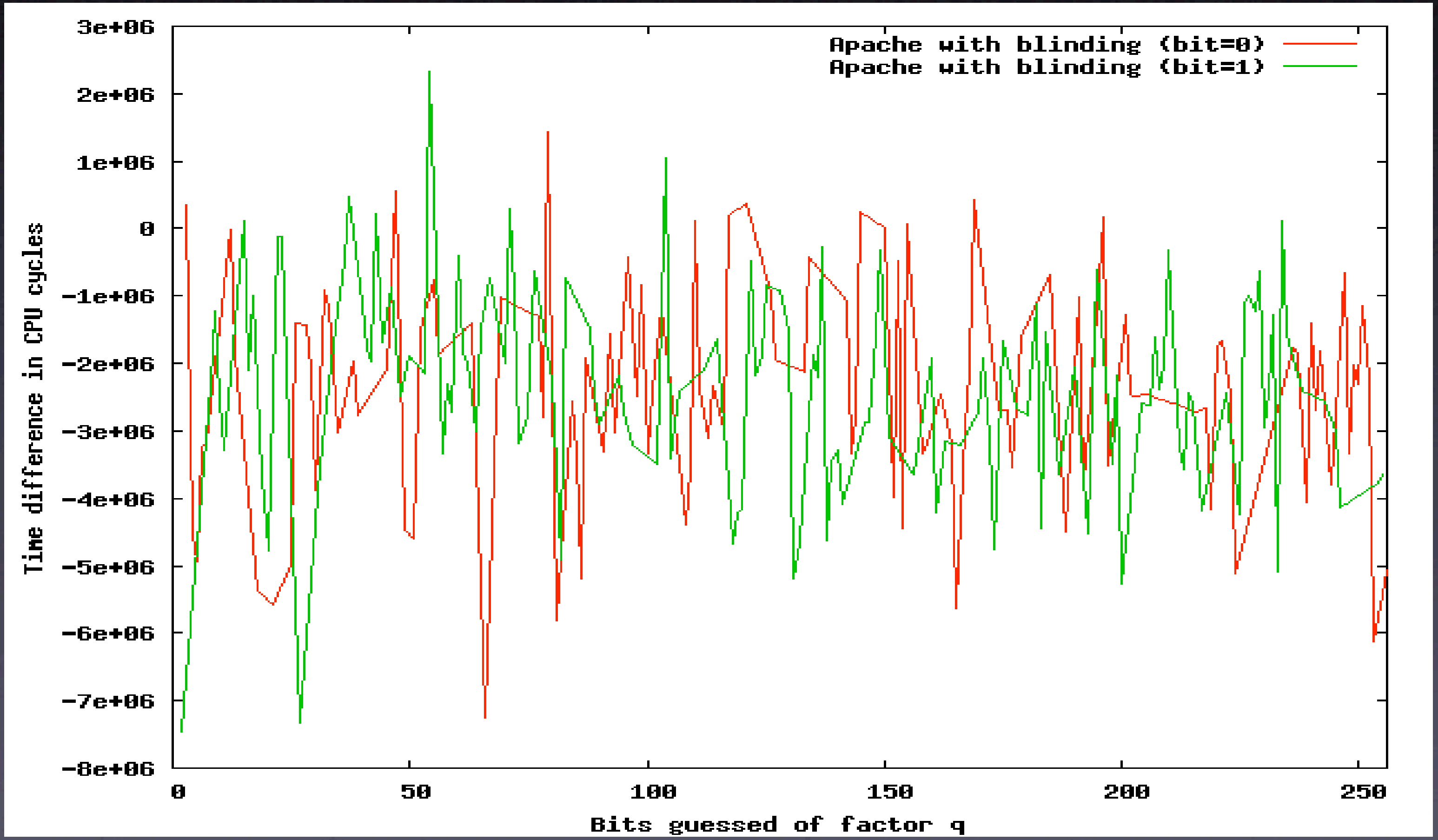
- Make running time independent of input
  - Montgomery: perform dummy reductions
  - Multiplication: always use Karatsuba (shifts)
- Make all operations take the same time

Countermeasures

- Blinding



Countermeasures



Blinding

- How do we know it prevents other attacks?
- Blinding is not provably secure
- What about template attacks?

Impact

- CERT advisory
- At least 37 products vulnerable
- 23 not vulnerable
- 56 unknown

Questions?

Montgomery Reduction

- $x \cdot y \bmod q \to x' \cdot y' \bmod 2^k$
- $2^k > q$ and $\gcd(2^k, q) = 1$
- Multiplication and division by powers of 2 is efficient

Karatsuba

- $A \times B = A_H A_L \times B_H B_L$

  $A \times B = (2^{\frac{n}{2}} A_H + A_L) \times (2^{\frac{n}{2}} B_H + B_L)$

  $A \times B = 2^n A_H B_H + 2^{\frac{n}{2}} (A_H B_L + A_L B_H) + A_L B_L$

Karatsuba

$$A \times B = 2^n A_H B_H + 2^{\frac{n}{2}} (A_H B_L + A_L B_H) + A_L B_L$$

$$A_H B_L + A_L B_H = (A_H + A_L) \times (B_H + B_L) - A_H B_H - A_L B_L$$

$$A \times B = 2^n A_H B_H + 2^{\frac{n}{2}} [(A_H + A_L) \times (B_H + B_L) - A_H B_H - A_L B_L] + A_L B_L$$

Karatsuba

- 3 multiplications and 2 shift and 7 additions
- multiplications fit in registers (no overflows)