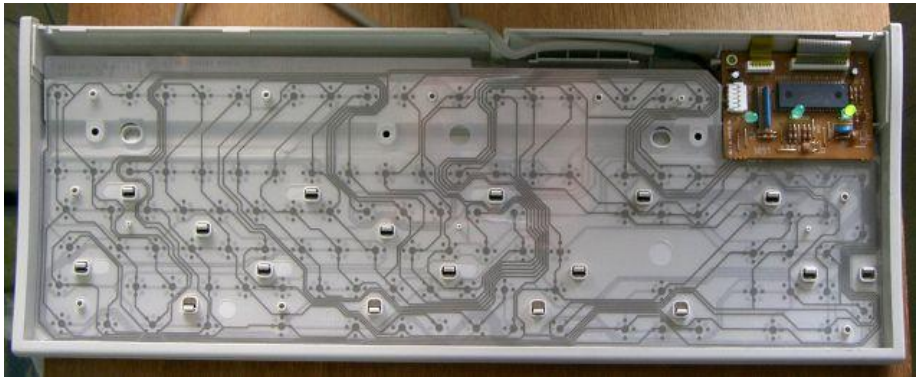
CS 6431

Side-Channel Attacks

Vitaly Shmatikov

Acoustic Information in Typing

- Different keystrokes make different sounds
  - Different locations on the supporting plate
  - Each key is slightly different



- Frequency information in the sound of the typed key can be used to learn which key it is
  - Observed by Asonov and Agrawal (2004)

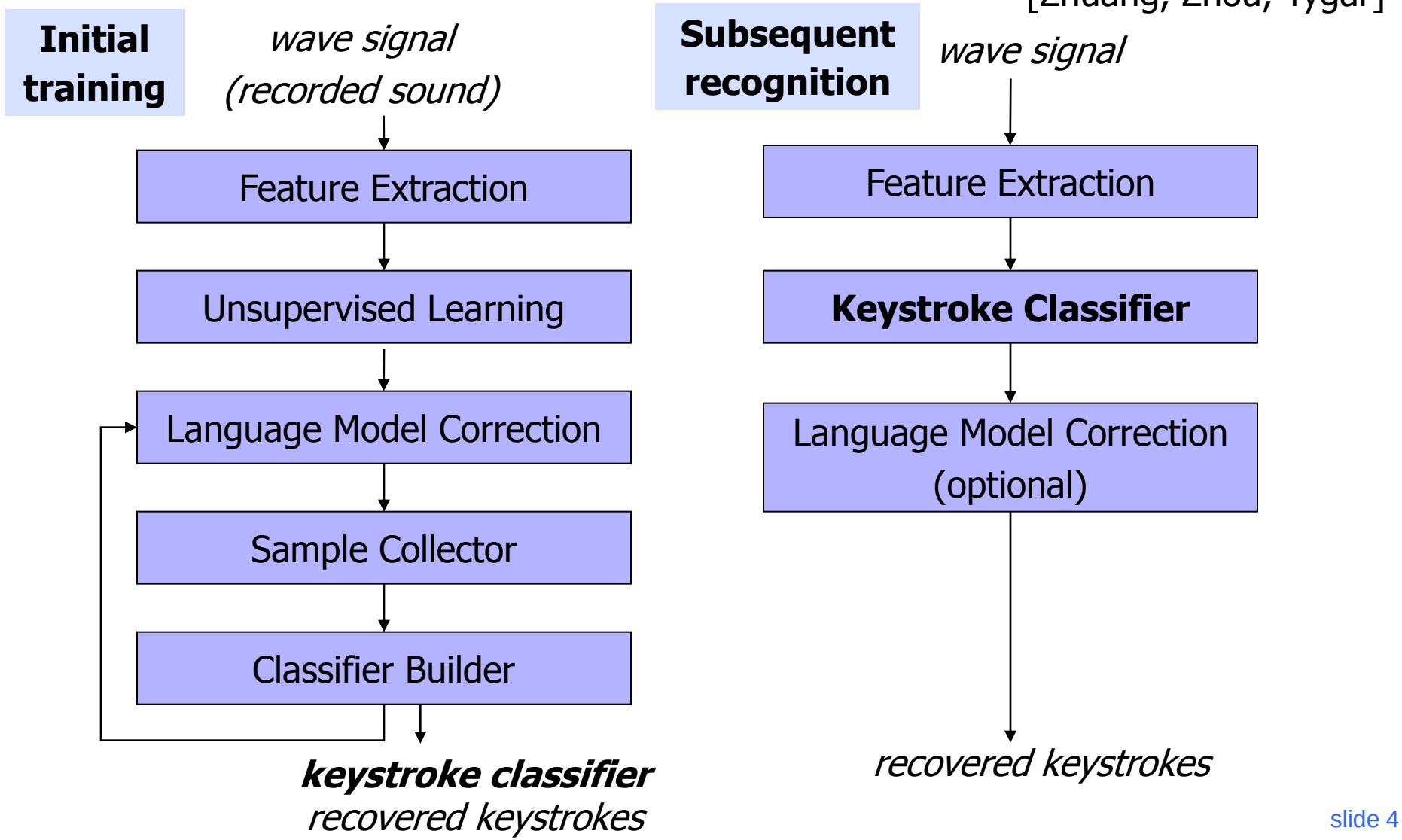
"Key" Observation

[Zhuang et al. "Keyboard Acoustic Emanations Revisited". CCS 2005]

- Build acoustic model for keyboard and typist
- Exploit the fact that typed text is non-random (for example, English)
  - Limited number of words
  - Limited letter sequences (spelling)
  - Limited word sequences (grammar)
- This requires a natural language model

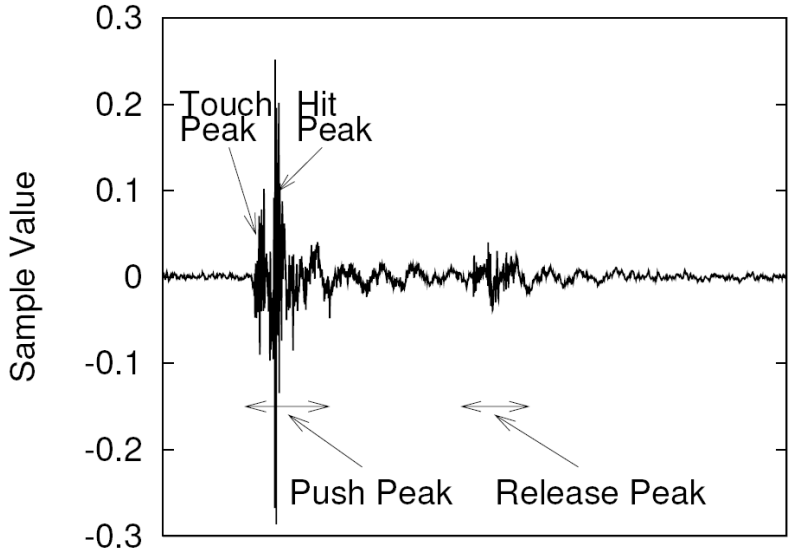
Overview

[Zhuang, Zhou, Tygar]



Sound of a Keystroke

[Zhuang, Zhou, Tygar]

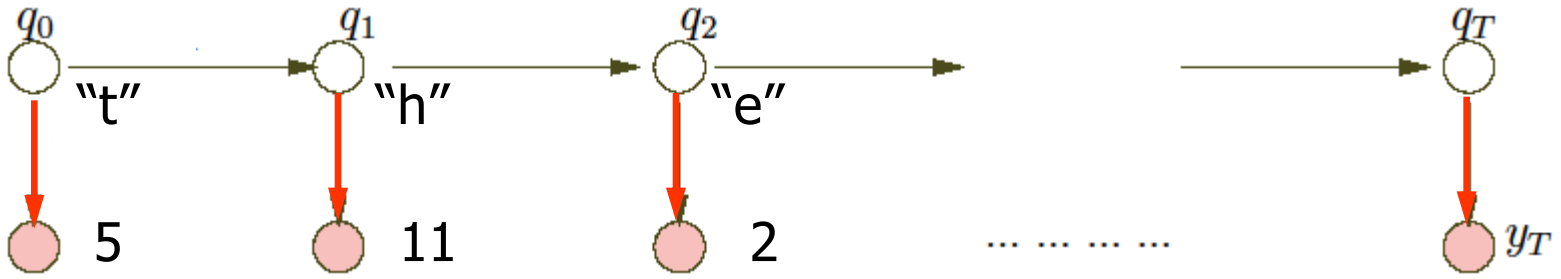


- Each keystroke is represented as a vector of Cepstrum features
  - Fourier transform of the decibel spectrum
  - Standard technique from speech processing

Bi-Grams of Characters

[Zhuang, Zhou, Tygar]

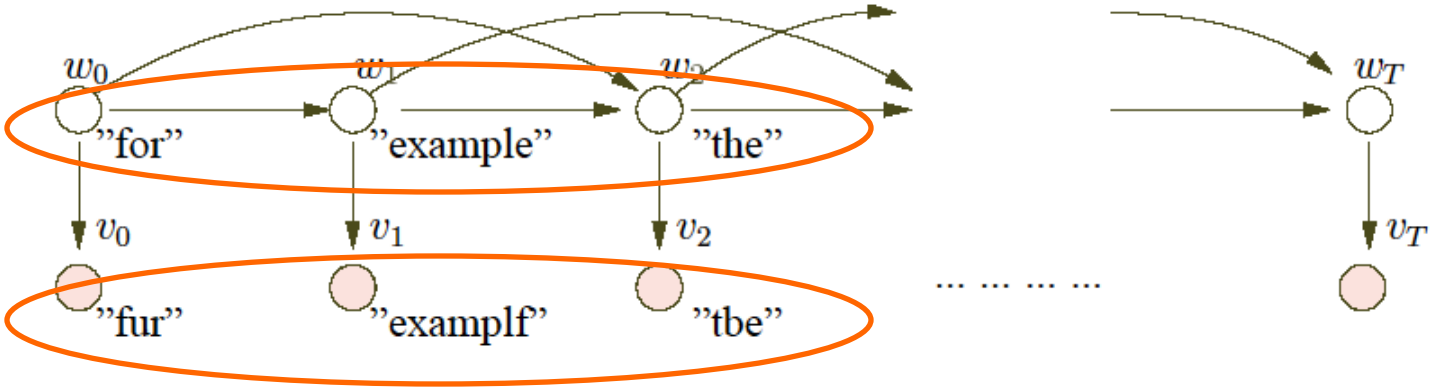
- Group keystrokes into N clusters
- Find the best mapping from cluster labels to characters
- Exploit the fact that some 2-character combinations are more common than others
  - Example: "th" vs. "tj"
  - Hidden Markov Models (HMM) for unsupervised learning



Spelling and Grammar

[Zhuang, Zhou, Tygar]

- First, spelling correction
- Then, simple statistical model of English grammar: common tri-grams of words
  - Use HMMs again to model



Recovered Text

[Zhuang, Zhou, Tygar]

Before spelling and grammar correction

the big money fight has drawn the shoporo od dosens of companies in the entertainment industry as well as attorneys gnnerals on states, who fear the fild shading softwate will encourage illegal acyivitt, srem the grosth of small arrists and lead to lost cobs and dimished sales tas revenue.

After spelling and grammar correction

the big money fight has drawn the support of dozens of companies in the entertainment industry as well as attorneys generals in states, who fear the film sharing software will encourage illegal activity, stem the growth of small artists and lead to lost jobs and finished sales tax revenue.

_____ = errors in recovery

◯ = errors corrected by grammar

Feedback-based Training

[Zhuang, Zhou, Tygar]

- Recovered characters + language correction provide feedback for more rounds of training
- Output: keystroke classifier
  - Language-independent
  - Can be used to recognize random sequences of keys
    - For example, passwords
  - Many possible representations of keystroke classifier
    - Neural networks, linear classification, Gaussian mixtures

Experiment: Single Keyboard

[Zhuang, Zhou, Tygar]

- Logitech Elite Duo wireless keyboard
- 4 data sets recorded in two settings: quiet and noisy
  - Consecutive keystrokes are clearly separable
- Automatically extract keystroke positions in the signal with some manual error correction

Results for a Single Keyboard

[Zhuang, Zhou, Tygar]

- Datasets

	Recording length	Number of words	Number of keys
Set 1	~12 min	~400	~2500
Set 2	~27 min	~1000	~5500
Set 3	~22 min	~800	~4200
Set 4	~24 min	~700	~4300

- Initial and final recognition rate

	Set 1 (%)		Set 2 (%)		Set 3 (%)		Set 4 (%)	
	Word	Char	Word	Char	Word	Char	Word	Char
Initial	35	76	39	80	32	73	23	68
Final	90	**96**	89	**96**	83	**95**	80	**92**

Experiment: Multiple Keyboards

[Zhuang, Zhou, Tygar]

- Keyboard 1: Dell QuietKey PS/2
  - In use for about 6 months

- Keyboard 2: Dell QuietKey PS/2
  - In use for more than 5 years

- Keyboard 3: Dell Wireless Keyboard
  - New

Results for Multiple Keyboards

[Zhuang, Zhou, Tygar]

- 12-minute recording with app. 2300 characters

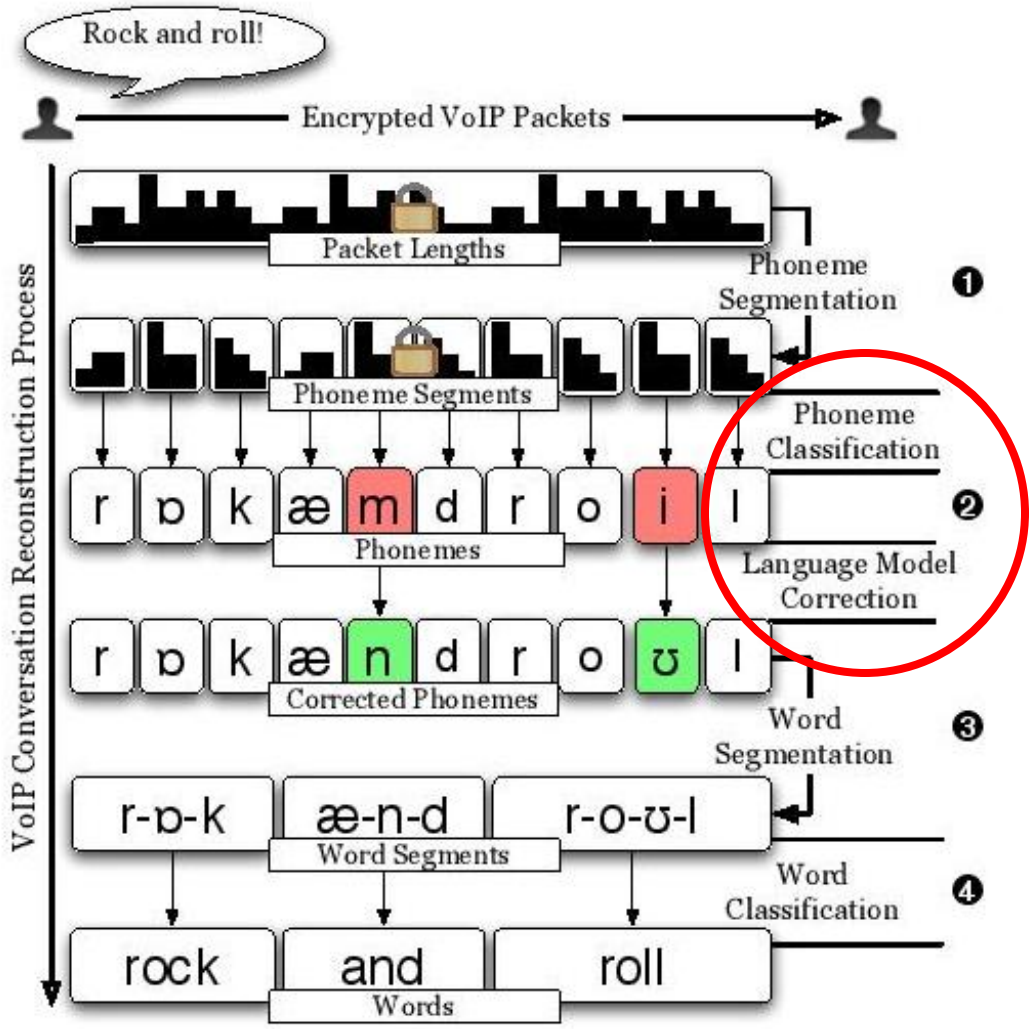
	Keyboard 1 (%)		Keyboard 2 (%)		Keyboard 3 (%)	
	Word	Char	Word	Char	Word	Char
Initial	31	72	20	62	23	64
Final	82	**93**	82	**94**	75	**90**

Encrypted Voice-over-IP

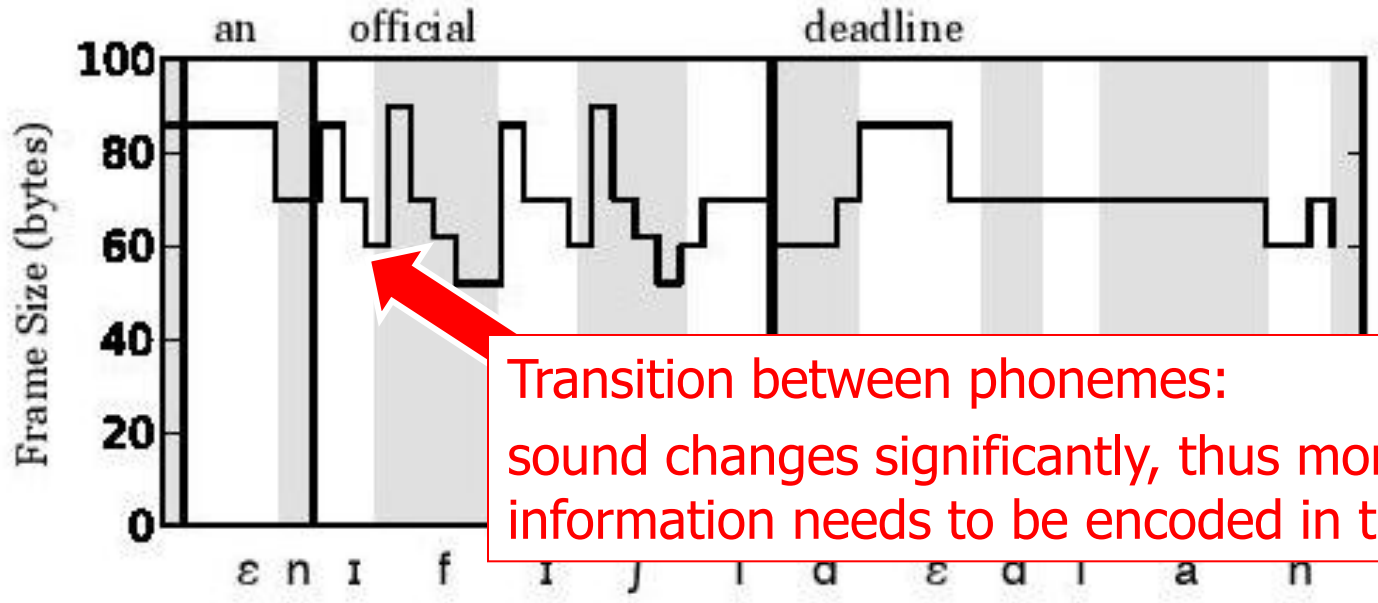
[White et al. "Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on fon-iks". Oakland 2011]

- Human speech = sequence of phonemes
  - Example: "rock and roll" = [ɹɒk ænd ɹoʊl]
- Variable-rate encoding: encodings of different phonemes have different lengths
- Stream ciphers preserve length of plaintext, therefore can observe length of individual encrypted phonemes
- Use a language model (phonotactics) to reconstruct speech without decrypting

Phonotactic Reconstruction



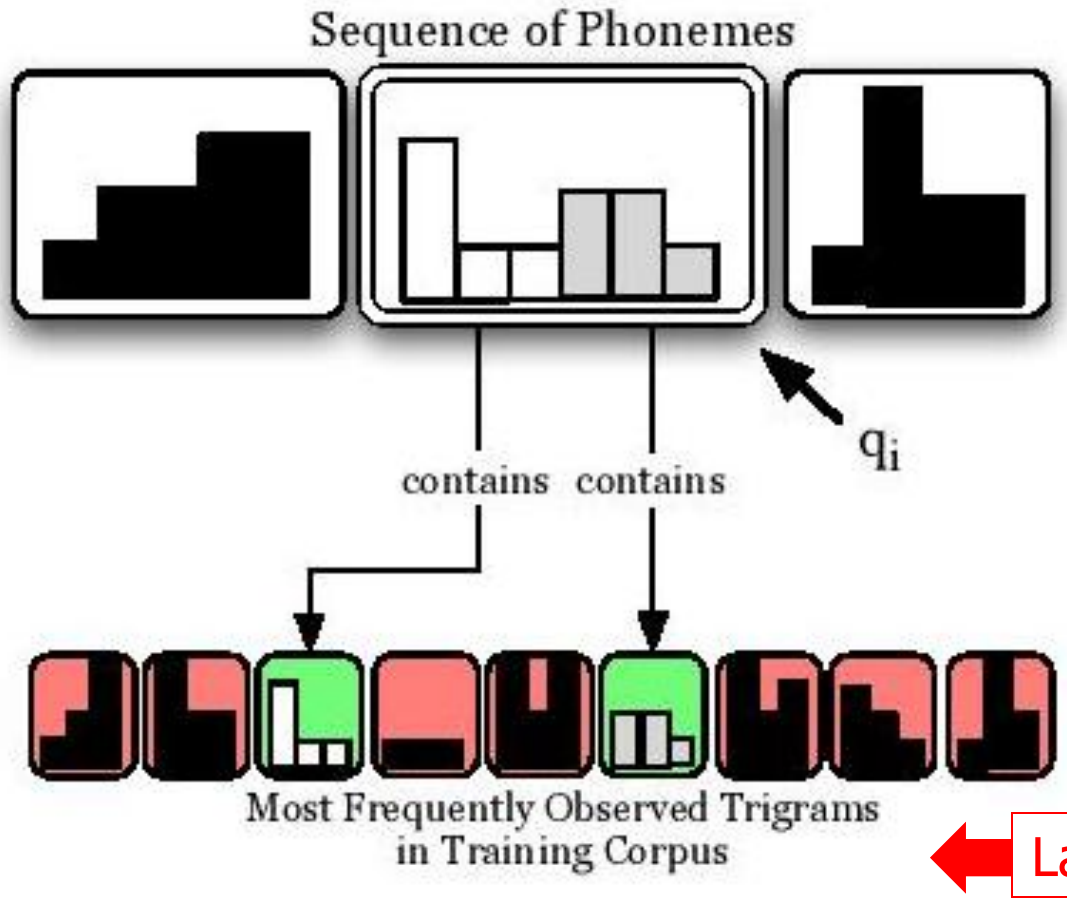
Segment Stream into Phonemes



Transition between phonemes:

sound changes significantly, thus more information needs to be encoded in the frame

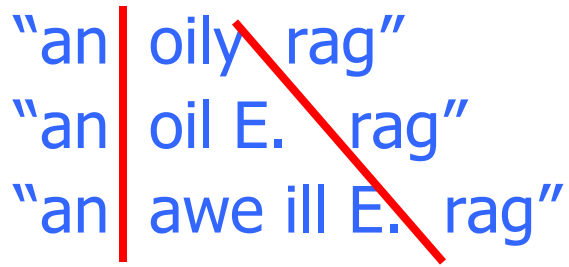
Frequency Analysis



Language correction

Segment Stream into Words

- Insert potential word breaks to break up phoneme triples that don't occur in any English word
- Match phonemes around the break with all possible phonemes that can end/begin words
- Use pronunciation dictionary to match phoneme subsequences to words, insert breaks at positions consistent with all matches
  - [inɔliɹæg]



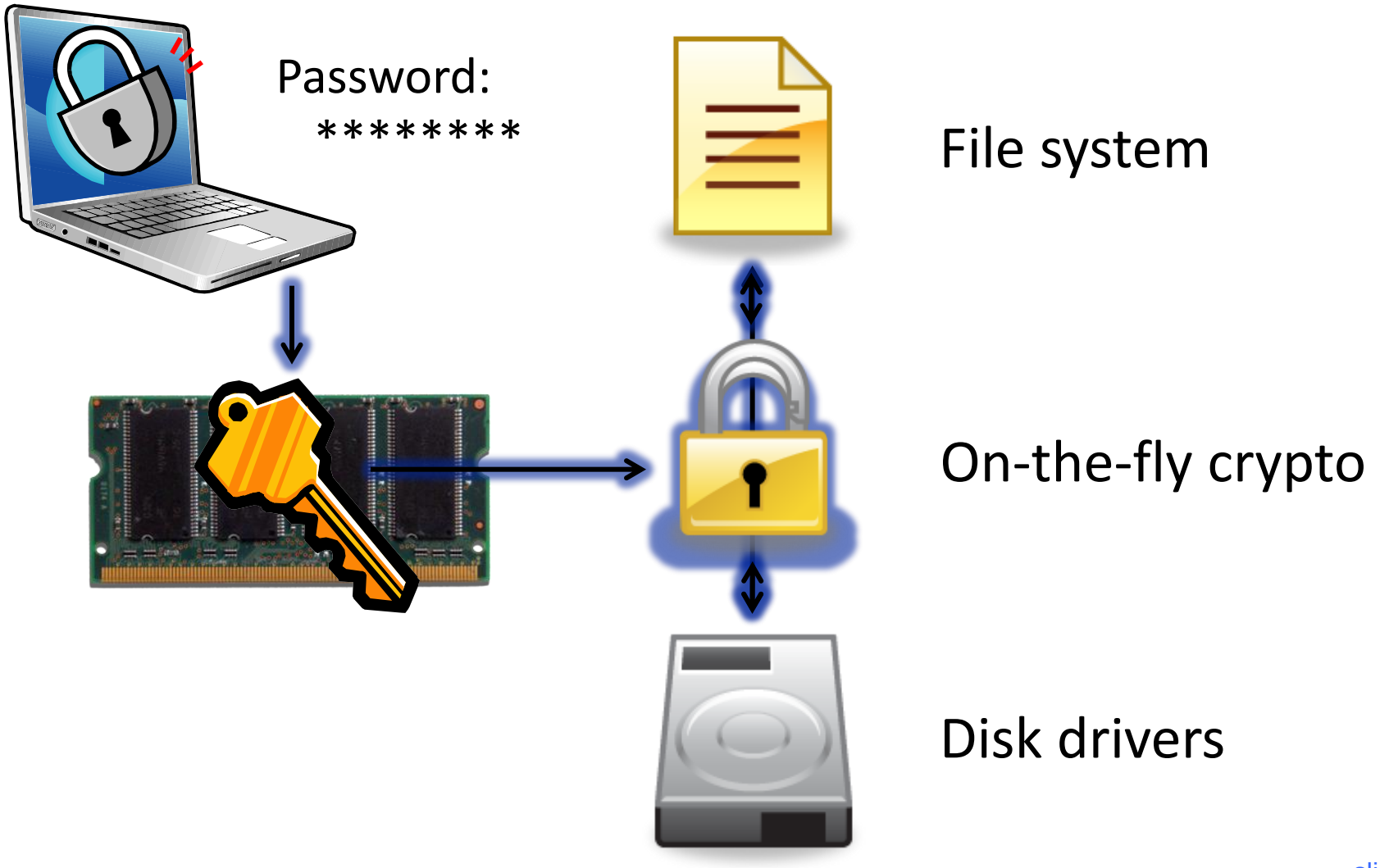
Reconstruct Words

- Previous step produces phoneme sequences corresponding to individual words
- Compute phonetic edit distance between each phoneme sequence and each word in the pronunciation dictionary
- Use word and part-of-speech model to disambiguate between homophones
  - "ate" vs. "eight"
- Result: human-understandable transcript of the conversation

Attacking Cryptographic Schemes

- Cryptanalysis
  - Find mathematical weaknesses in constructions
  - Statistical analysis of plaintext / ciphertext pairs
- Side-channel attacks
  - Exploit characteristics of implementations
  - Power analysis
  - Electromagnetic radiation analysis
  - Acoustic analysis
  - Timing analysis

Protecting Data on a Laptop

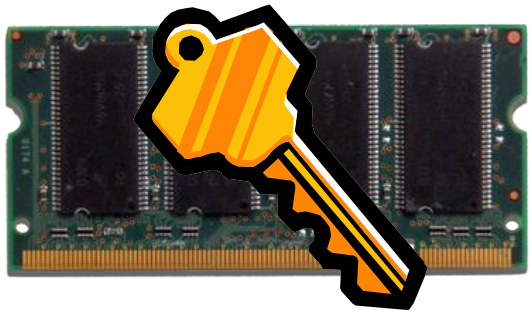


Common Attack Scenario

Security Assumptions

The encryption is strong

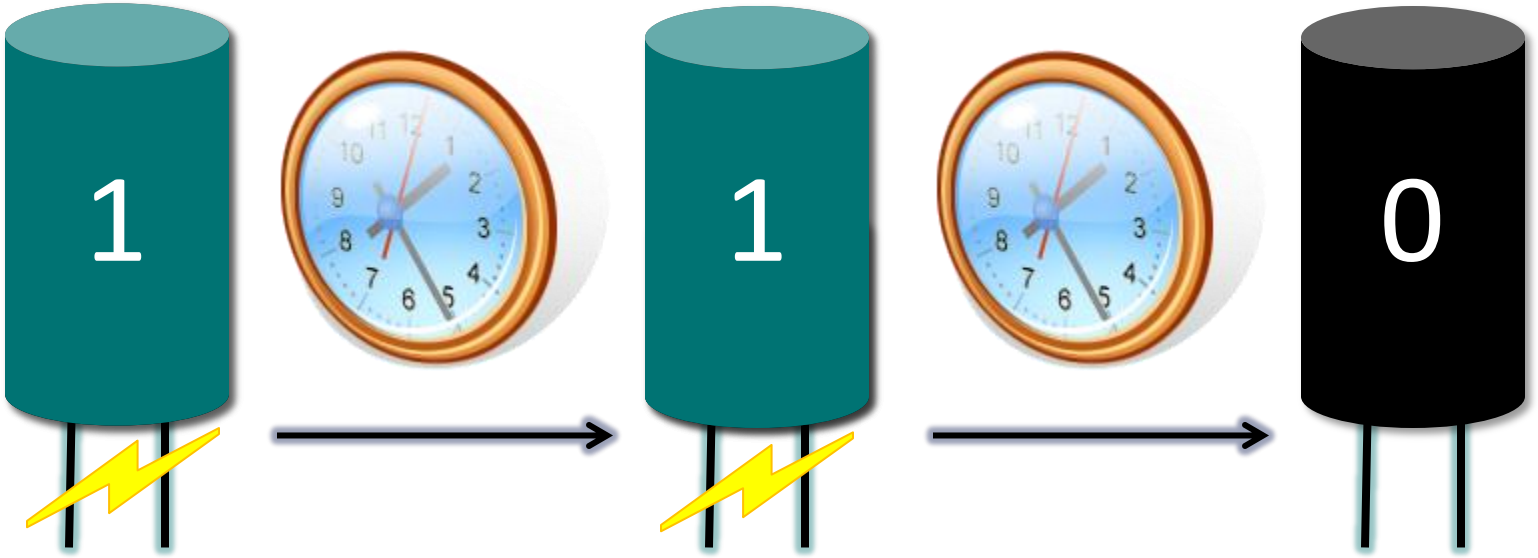
The OS protects the key in RAM



…the attacker might reboot to circumvent the OS, but since RAM is volatile, the key will be lost…

Dynamic RAM Volatility

DRAM cell
(capacitor)



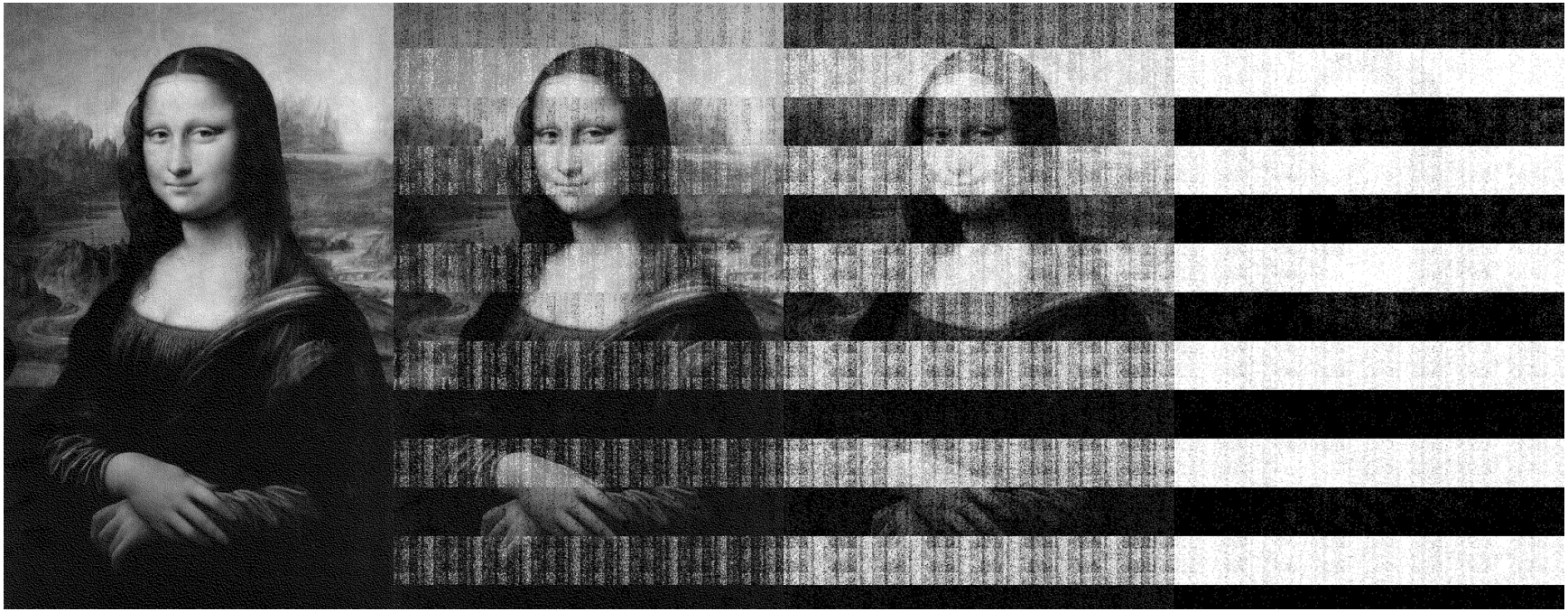
Write “1”

Refresh (read and rewrite)

Refresh interval ≈ 32 ms

What if we don’t refresh?

Decay After Cutting Power

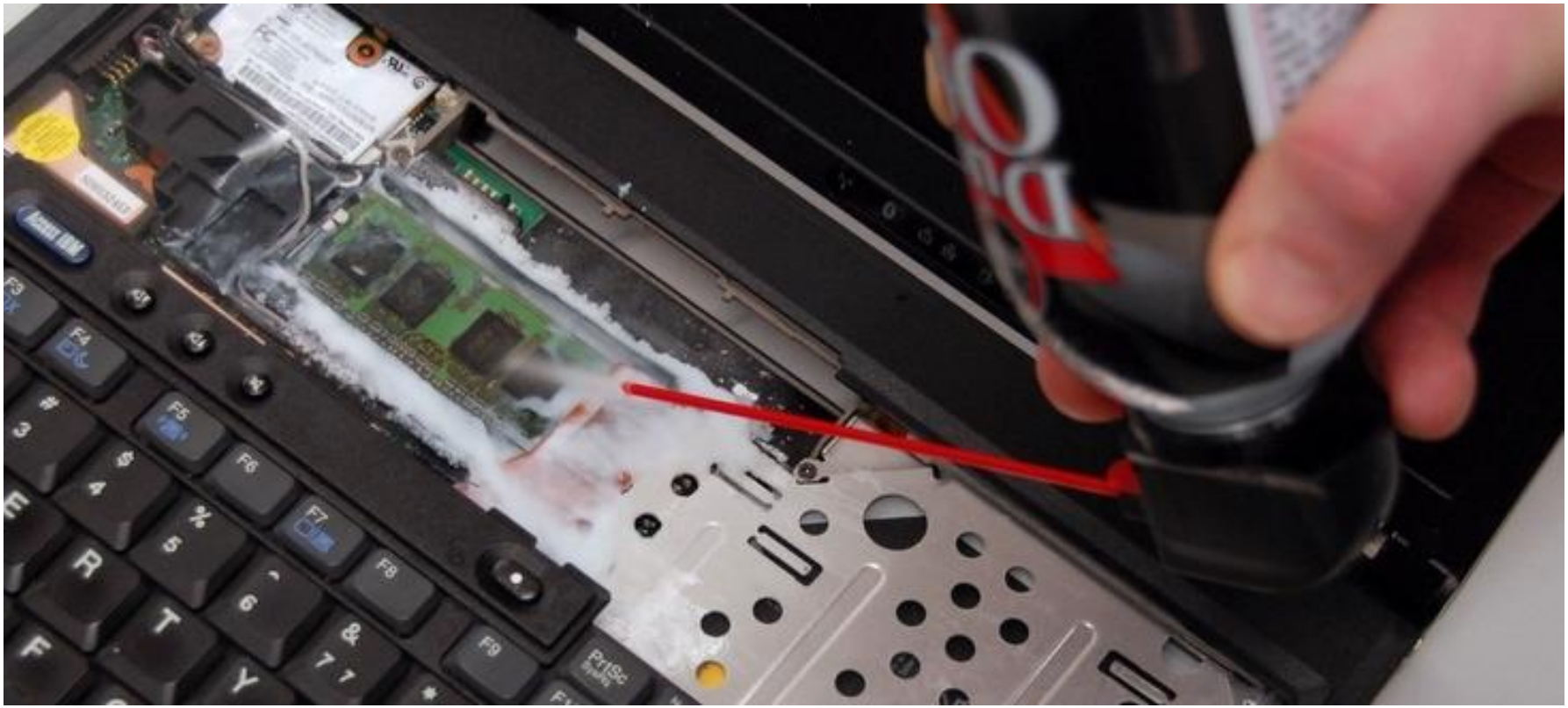


5 secs 30 secs 60 secs 300 secs

Capturing Residual Data

- No special equipment needed, but …
- … booting OS overwrites large areas of RAM
- Solution: boot a small low-level program to dump out memory contents
  - PXE dump (9 KB)
  - EFI dump (10 KB)
  - USB dump (22 KB)
- What if BIOS clears RAM?
  - Common on systems with error-corrected RAM

Slowing Decay By Cooling



-50°C

< 0.2% decay after **1 minute**


Esc
F1
F2
F3
Tab
Caps
START/STOP
SET
MODE
SPLIT/RESET
SELECT
6:48
29
GOWalkin
by SPORT

Even Cooler

Liquid nitrogen -196°C

< 0.17% decay after **1 hour**

Not necessary in practice

Dealing with Bit Errors

[Halderman et al. "Cold Boot Attacks on Encryption Keys". USENIX Security 2008]

Some bit errors inevitable, especially without cooling (increasing with memory density)

Naïve Approach

Given corrupted K', find K:

Brute-force search over low Hamming distance to K'

e.g. 256-bit key with 10% error:

> 2^{56} guesses (too slow!)

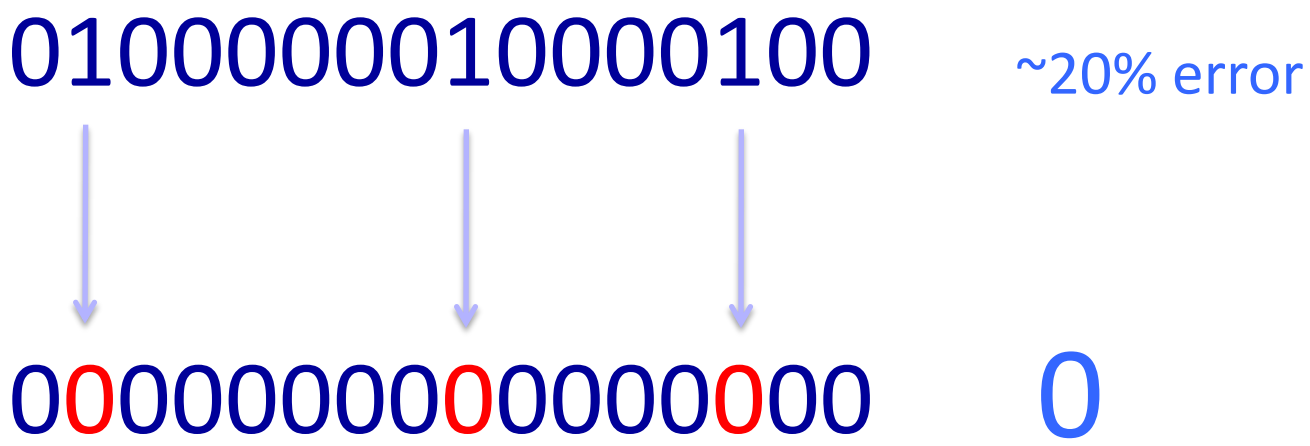
Insight

Most programs store precomputed derivatives of K (e.g., key schedules)

These derivatives contain redundancy; can treat them as error correcting codes

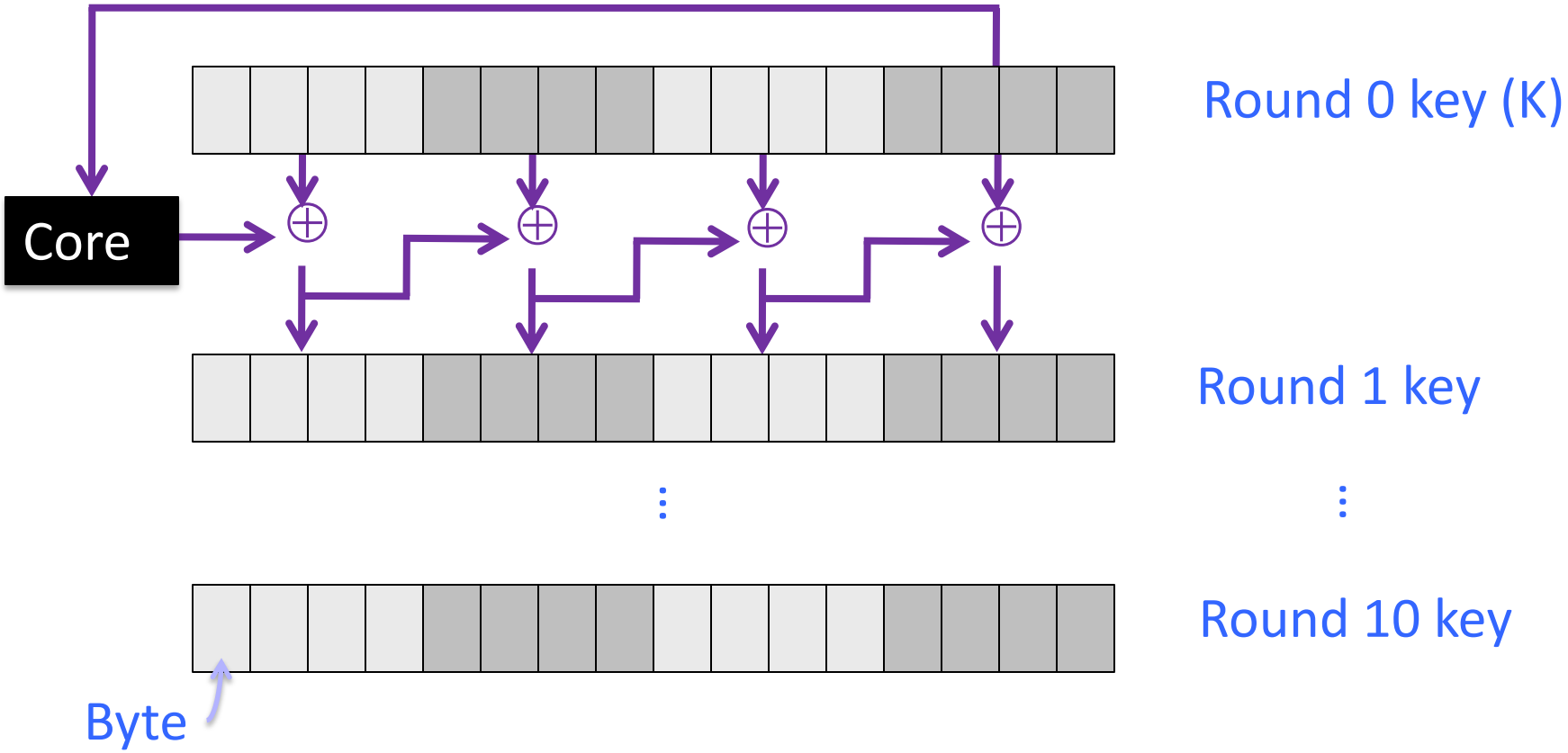
Correcting Bit Errors in DES

Key schedule contains ~14 redundant copies of each bit from the key



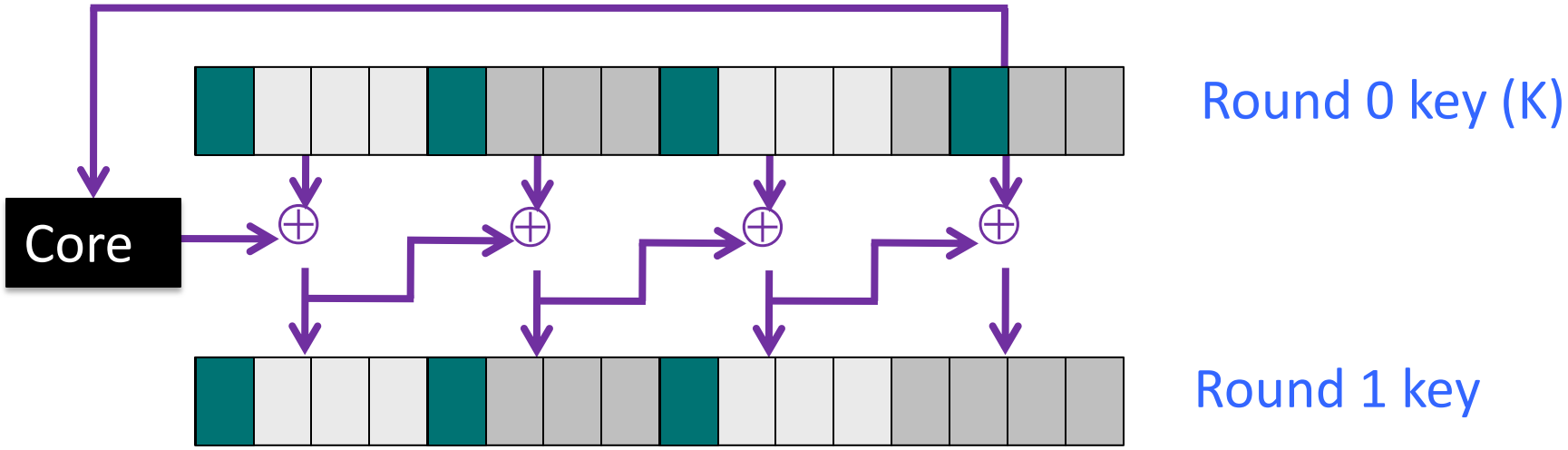
AES Key Schedule

128-bit key K → 10 more 128-bit keys for cipher rounds



Correcting Bit Errors in AES (1)

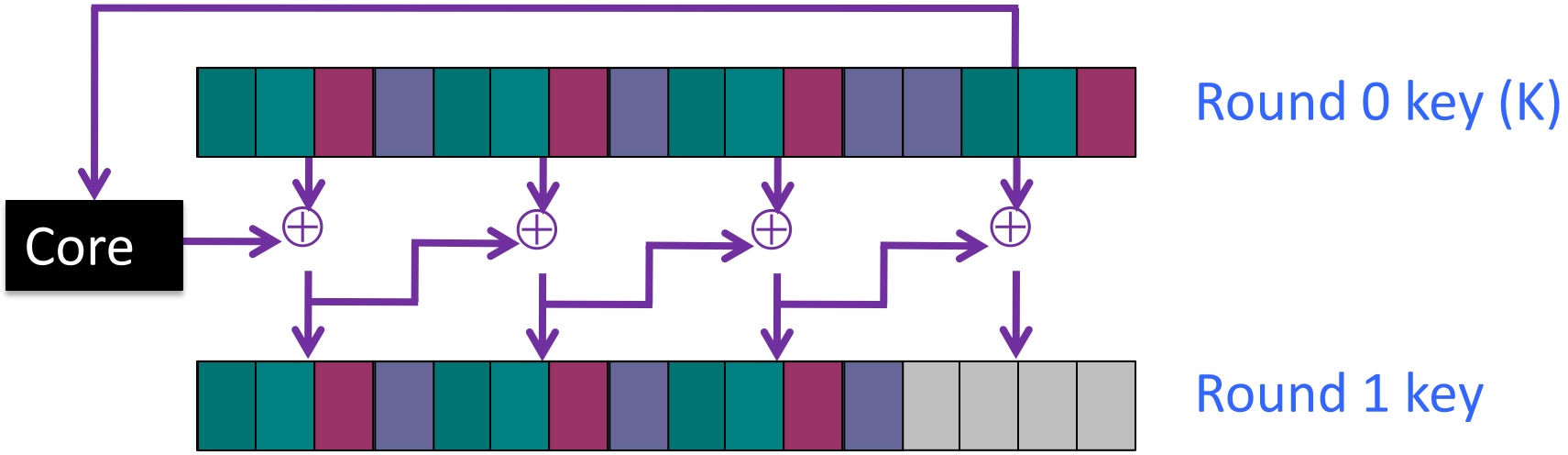
Key schedule recovered from memory (contains errors)



1. Slices: 4 bytes in Round 0 determine 3 bytes in Round 1
2. Enumerate 2^{32} possibilities for each 7 byte slice
3. Eliminate values unlikely to have decayed to observed bytes (excludes vast majority)

Correcting Bit Errors in AES (2)

Key schedule recovered from memory (contains errors)



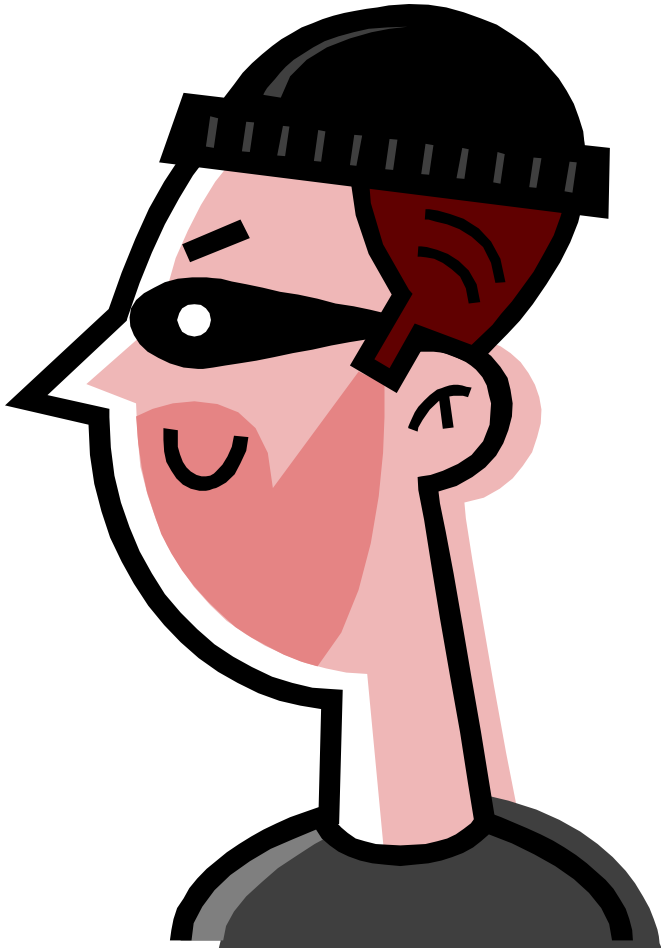
4. Repeat for each of the 4 slices
5. Combine possible slice values into candidate keys
6. Test candidates keys by expanding them into full key schedules – compare to recovered memory

Finding AES Key Schedules

- Iterate through each byte of memory
- Treat following region as an AES key schedule
- For each word in the candidate "schedule":
  - Calculate correct value, assuming other bytes correct
  - Take Hamming distance to observed value
- If total distance is low, output the key

Demonstrated Attacks

- Windows BitLocker
- Mac OS FileVault
- Linux dm-crypt
- Linux LoopAES
- TrueCrypt



Timing Attack

- Basic idea: learn the system's secret by observing how long it takes to perform various computations
- Typical goal: extract private key
- Extremely powerful because isolation doesn't help
  - Victim could be remote
  - Victim could be inside its own virtual machine
  - Keys could be in tamper-proof storage or smartcard
- Attacker wins simply by measuring response times

RSA Cryptosystem

- Key generation:
  - Generate large (say, 512-bit) primes p, q
  - Compute n=pq and $\varphi(n)=(p-1)(q-1)$
  - Choose small e, relatively prime to $\varphi(n)$
    - Typically, e=3 (may be vulnerable) or $e=2^{16}+1=65537$ (why?)
  - Compute unique d such that $ed = 1 \bmod \varphi(n)$
  - Public key = (e,n); private key = d
    - Security relies on the assumption that it is difficult to compute roots modulo n without knowing p and q
- Encryption of m (simplified!): $c = m^e \bmod n$
- Decryption of c: $c^d \bmod n = (m^e)^d \bmod n = m$

How RSA Decryption Works

- RSA decryption: compute y^x mod n
  - This is a modular exponentiation operation
- Naïve algorithm: square and multiply

```
Let s_0 = 1.
For k = 0 upto w − 1:
  If (bit k of x) is 1 then
    Let R_k = (s_k · y) mod n.
  Else
    Let R_k = s_k.
  Let s_{k+1} = R_k^2 mod n.
EndFor.
Return (R_{w−1}).
```

Kocher's Observation

Let $s_0 = 1$.
For $k = 0$ upto $w - 1$:
  If (bit k of x) is 1 then
    Let $R_k = (s_k \cdot y) \bmod n$.
  Else
    Let $R_k = s_k$.
  Let $s_{k+1} = R_k^2 \bmod n$.
EndFor.
Return (R_{w-1}).

Whether iteration takes a long time depends on the k^{th} bit of secret exponent

This takes a while to compute

This is instantaneous

Exploiting Timing Dependency

- Assumption / heuristic: timings of subsequent multiplications are independent
  - Given that we know the first k-1 bits of x ...

    Given a guess for the k^{th} bit of x ...

    ... Time for remaining bits independent

Exact timing

Exact guess

- Given measurement of total time, determine whether there is correlation between "time for k^{th} bit is long" and "total time is long"

Outline of Kocher's Attack

- Idea: guess some bits of the exponent and predict how long decryption will take
- If guess is correct, will observe correlation; if incorrect, then prediction will look random
  - This is a signal detection problem, where signal is timing variation due to guessed exponent bits
  - The more bits you already know, the stronger the signal, thus easier to detect (error-correction property)
- Start by guessing a few top bits, look at correlations for each guess, pick the most promising candidate and continue

RSA in OpenSSL

[Brumley and Boneh.
"Remote Timing Attacks are Practical".
USENIX Security 2003]

- OpenSSL is a popular open-source toolkit
  - mod_SSL (in Apache = 28% of HTTPS market)
  - stunnel (secure TCP/IP servers)
  - sNFS (secure NFS)
  - Many more applications
- Kocher's attack doesn't work against OpenSSL
  - Instead of square-and-multiply, OpenSSL uses CRT, sliding windows and two different multiplication algorithms for modular exponentiation
    - CRT = Chinese Remainder Theorem
    - Secret exponent is processed in chunks, not bit-by-bit

Chinese Remainder Theorem

- $n = n_1 n_2 \ldots n_k$

  where $\gcd(n_i, n_j) = 1$ when $i \neq j$

- The system of congruences

  $x = x_1 \bmod n_1 = \ldots = x_k \bmod n_k$

  - Has a simultaneous solution x to all congruences
  - There exists exactly one solution x between 0 and n-1

- For RSA modulus n=pq, to compute x mod n it's enough to know x mod p and x mod q

RSA Decryption With CRT

- To decrypt c, need to compute $m=c^d \bmod n$
- Use Chinese Remainder Theorem (why?)
  - $d_1 = d \bmod (p-1)$
  - $d_2 = d \bmod (q-1)$
  - $qinv = q^{-1} \bmod p$

  } these are precomputed

  - Compute $m_1 = c^{d1} \bmod p$; $m_2 = c^{d2} \bmod q$
  - Compute $m = m_2+(qinv*(m_1-m_2) \bmod p)*q$

Attack this computation in order to learn q.
This is enough to learn private key **(why?)**

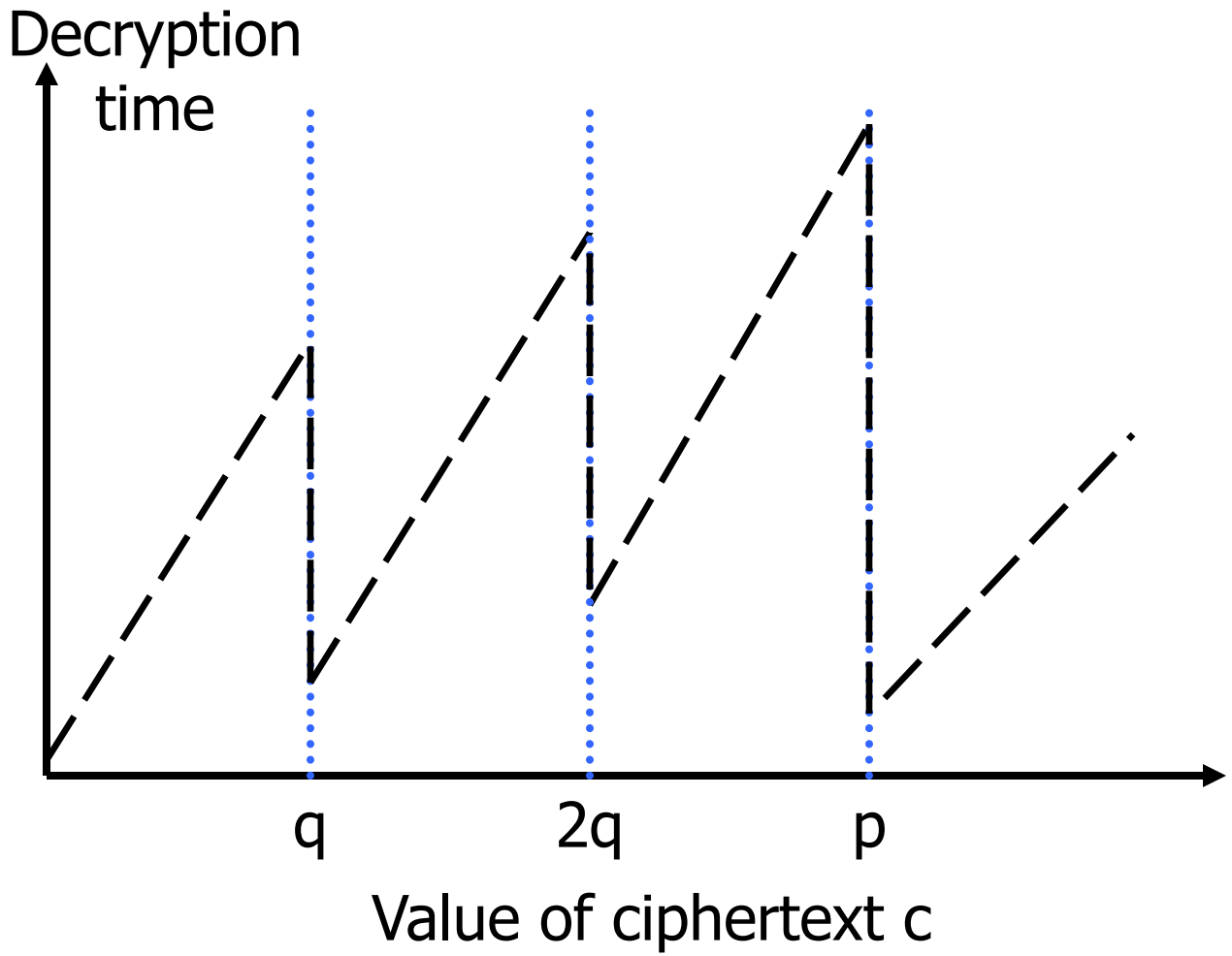
Montgomery Reduction

- Decryption requires computing $m_2 = c^{d2} \bmod q$
- This is done by repeated multiplication
  - Simple: square and multiply (process d_2 1 bit at a time)
  - More clever: sliding windows (process d_2 in 5-bit blocks)
- In either case, many multiplications modulo q
- Multiplications use Montgomery reduction
  - Pick some $R = 2^k$
  - To compute x*y mod q, convert x and y into their Montgomery form xR mod q and yR mod q
  - Compute $(xR * yR) * R^{-1} = zR \bmod q$
    - Multiplication by R^{-1} can be done very efficiently

Schindler's Observation

- At the end of Montgomery reduction, if zR > q, then need to subtract q
  - Probability of this extra step is proportional to c mod q
- If c is close to q, a lot of subtractions will be done
- If c mod q = 0, very few subtractions
  - Decryption will take longer as c gets closer to q, then become fast as c passes a multiple of q
- By playing with different values of c and observing how long decryption takes, attacker can guess q
  - Doesn't work directly against OpenSSL because of sliding windows and two multiplication algorithms

Reduction Timing Dependency

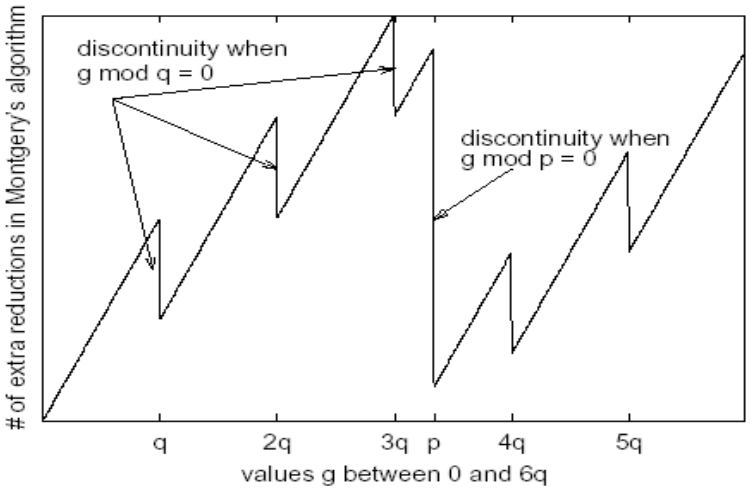


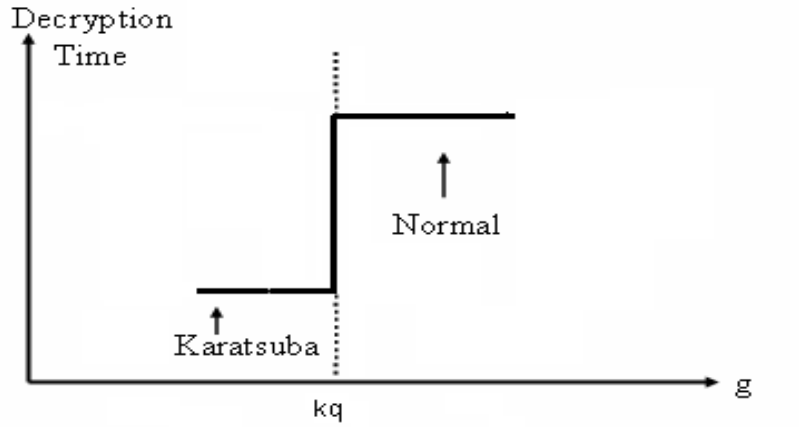
Integer Multiplication Routines

- 30-40% of OpenSSL running time is spent on integer multiplication
- If integers have the same number of words n, OpenSSL uses Karatsuba multiplication
  - Takes $O(n^{\log_2 3})$
- If integers have unequal number of words n and m, OpenSSL uses normal multiplication
  - Takes $O(nm)$

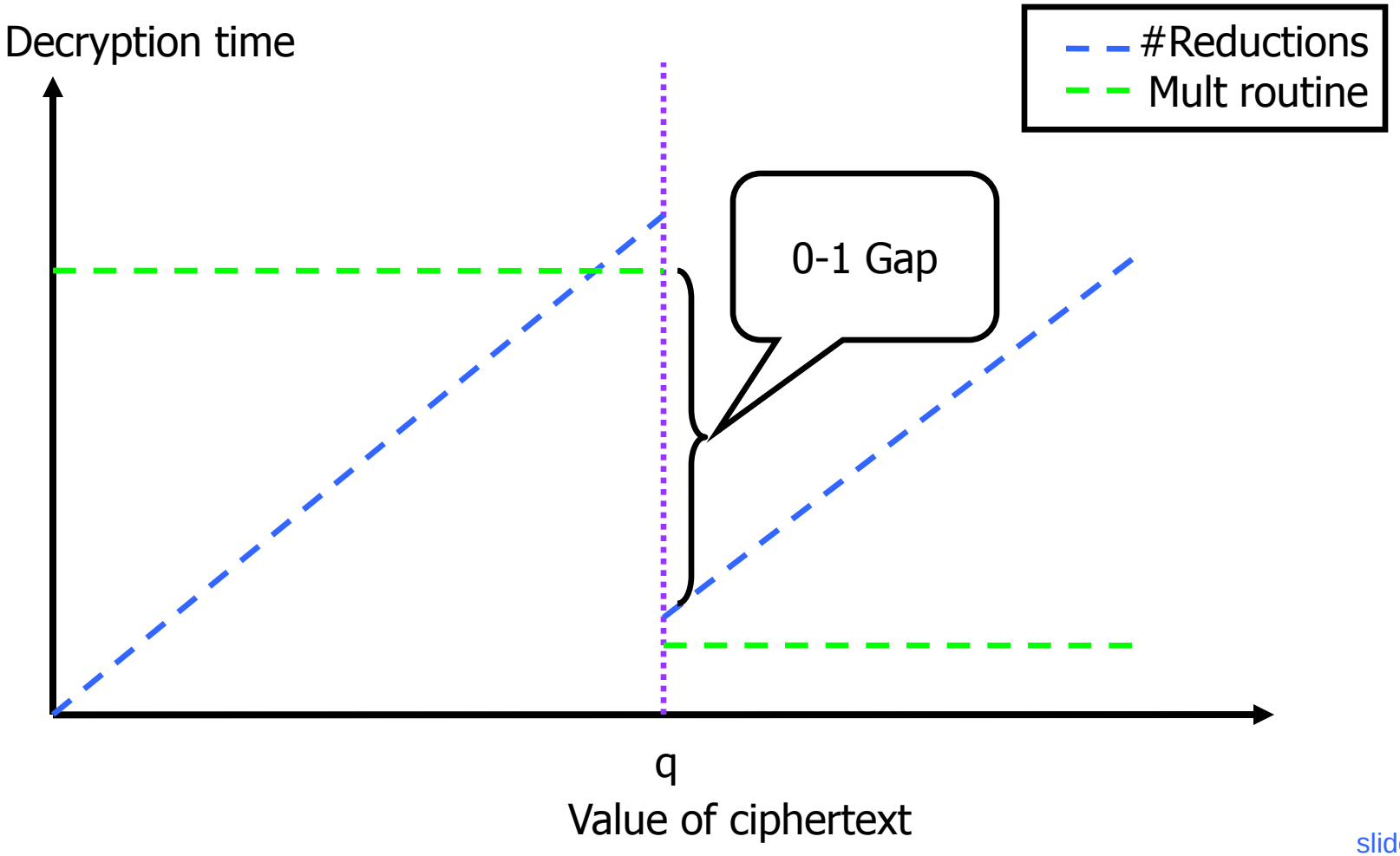
Summary of Time Dependencies

	g<q	g>q
Montgomery effect	Longer	Shorter
Multiplication effect	Shorter	Longer
g is the decryption value (same as c) Different effects… but one will always dominate!		

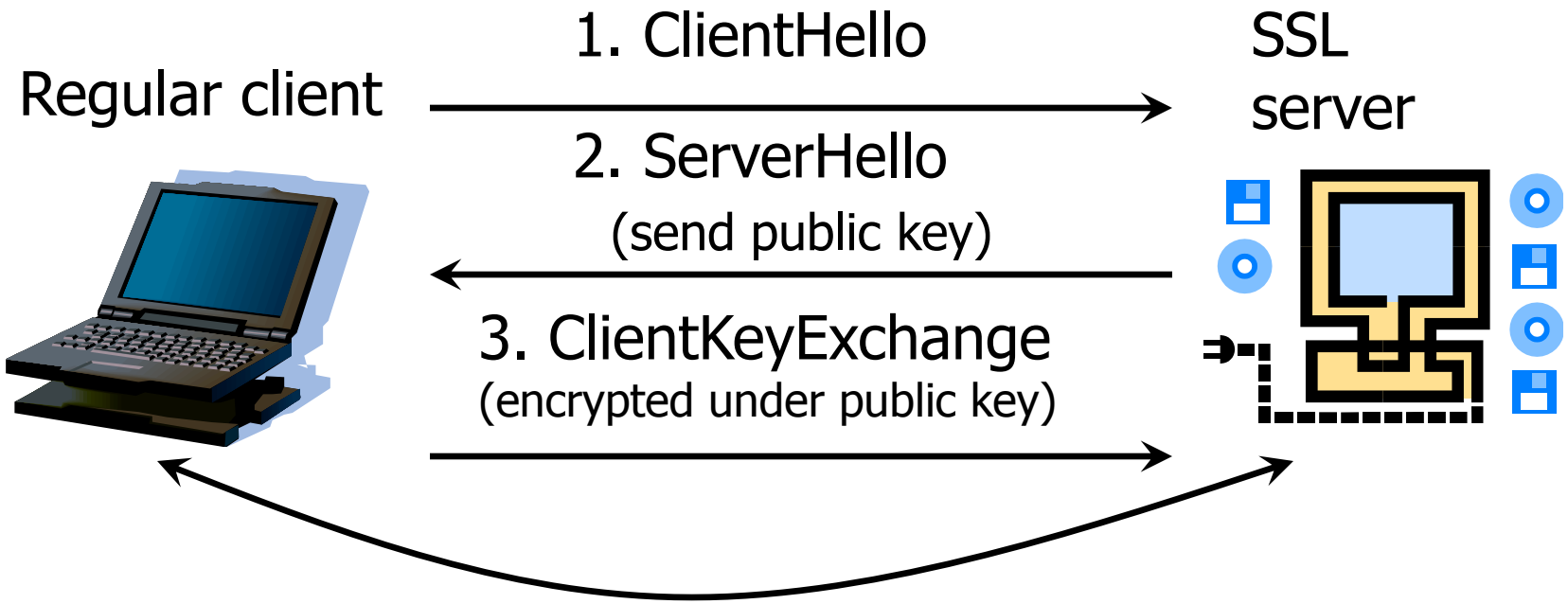




Discontinuity in Decryption Time



Normal SSL Handshake



Attacking SSL Handshake

Attacker

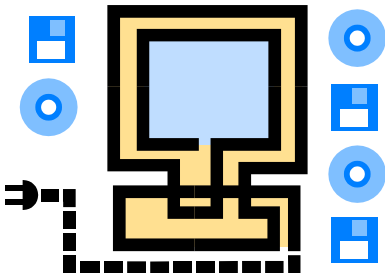
1. ClientHello

2. ServerHello
(send public key)

3. Record time t_1
Send guess g or g_{hi}

4. Alert

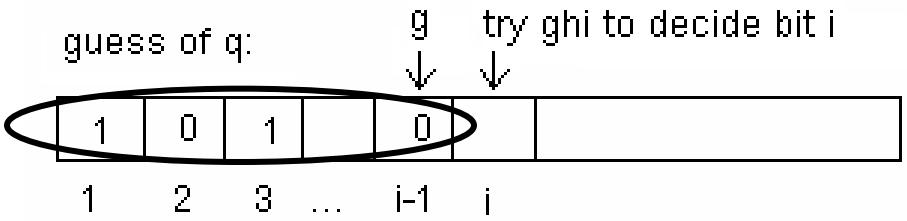
5. Record time t_2
Compute $t_2 - t_1$



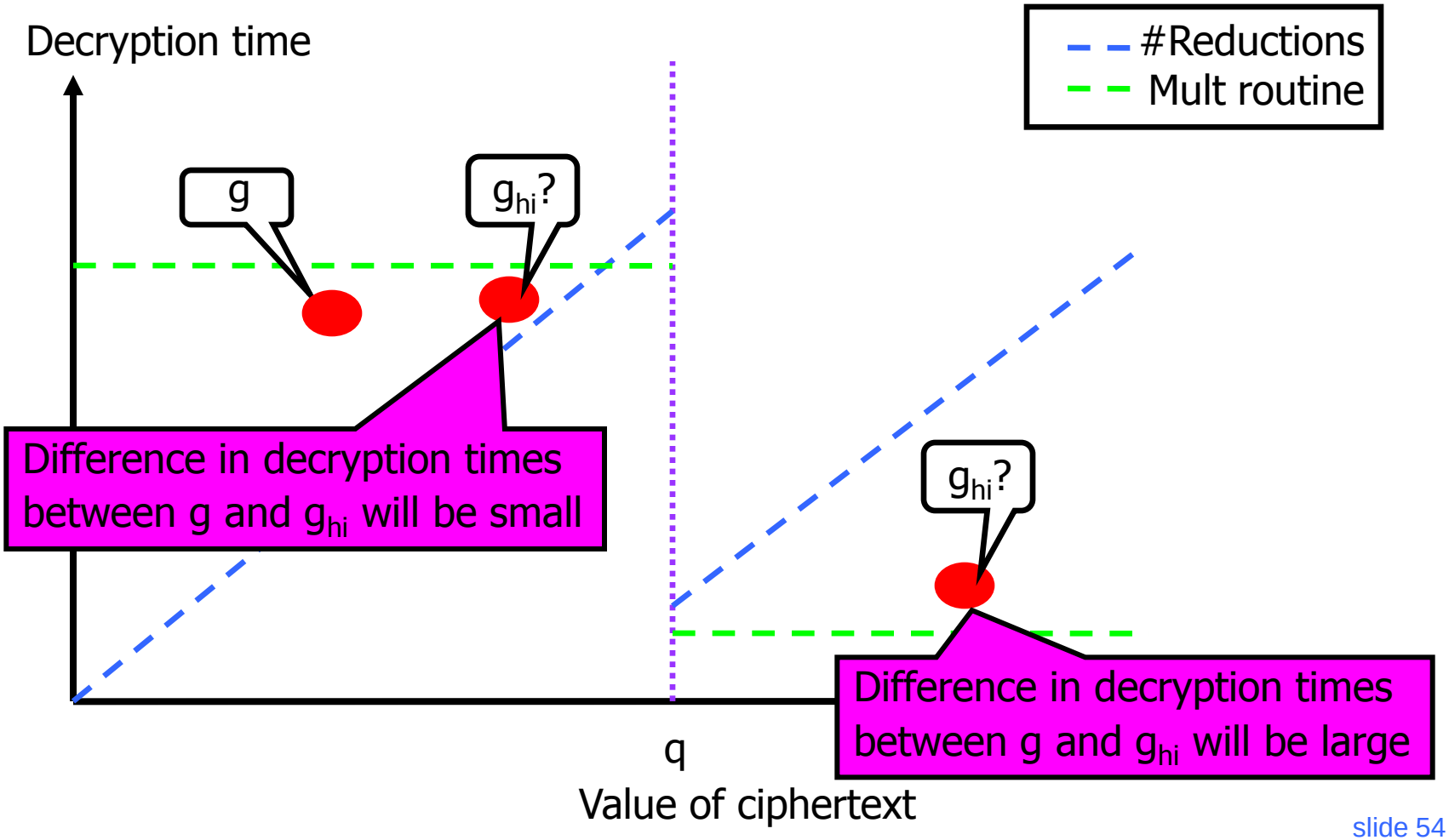
SSL server

Attack Overview

- Initial guess g for q between 2^{511} and 2^{512} (why?)
- Try all possible guesses for the top few bits
- Suppose we know i-1 top bits of q. Goal: i^{th} bit.
  - Set g =<known i-1 bits of q>000000
  - Set g_{hi}=<known i-1 bits of q>100000 - note: g<g_{hi}
    - If g<q<g_{hi} then the i^{th} bit of q is 0
    - If g<g_{hi}<q then the i^{th} bit of q is 1
- Goal: decide whether g<q<g_{hi} or g<g_{hi}<q



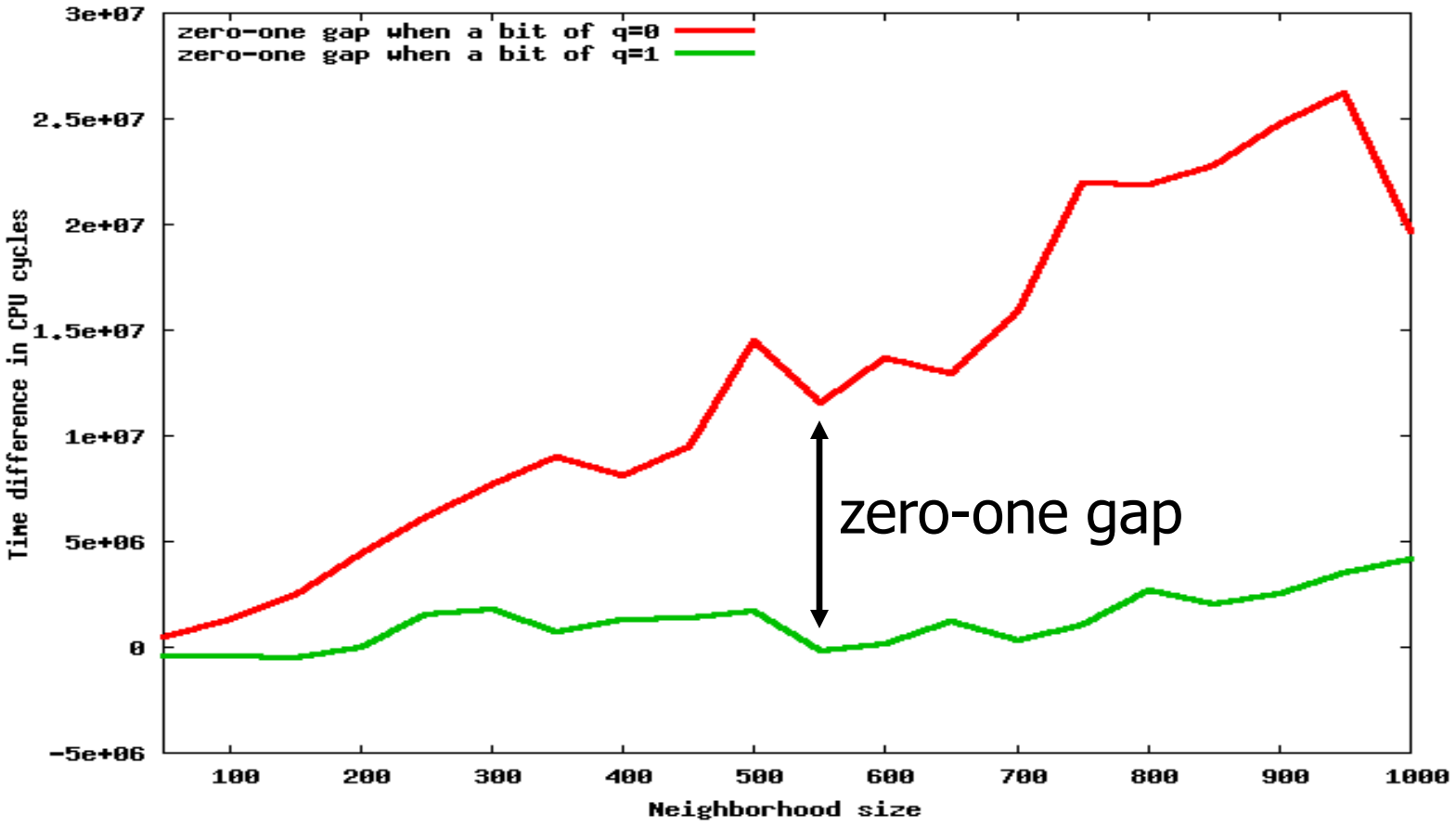
Two Possibilities for g_{hi}



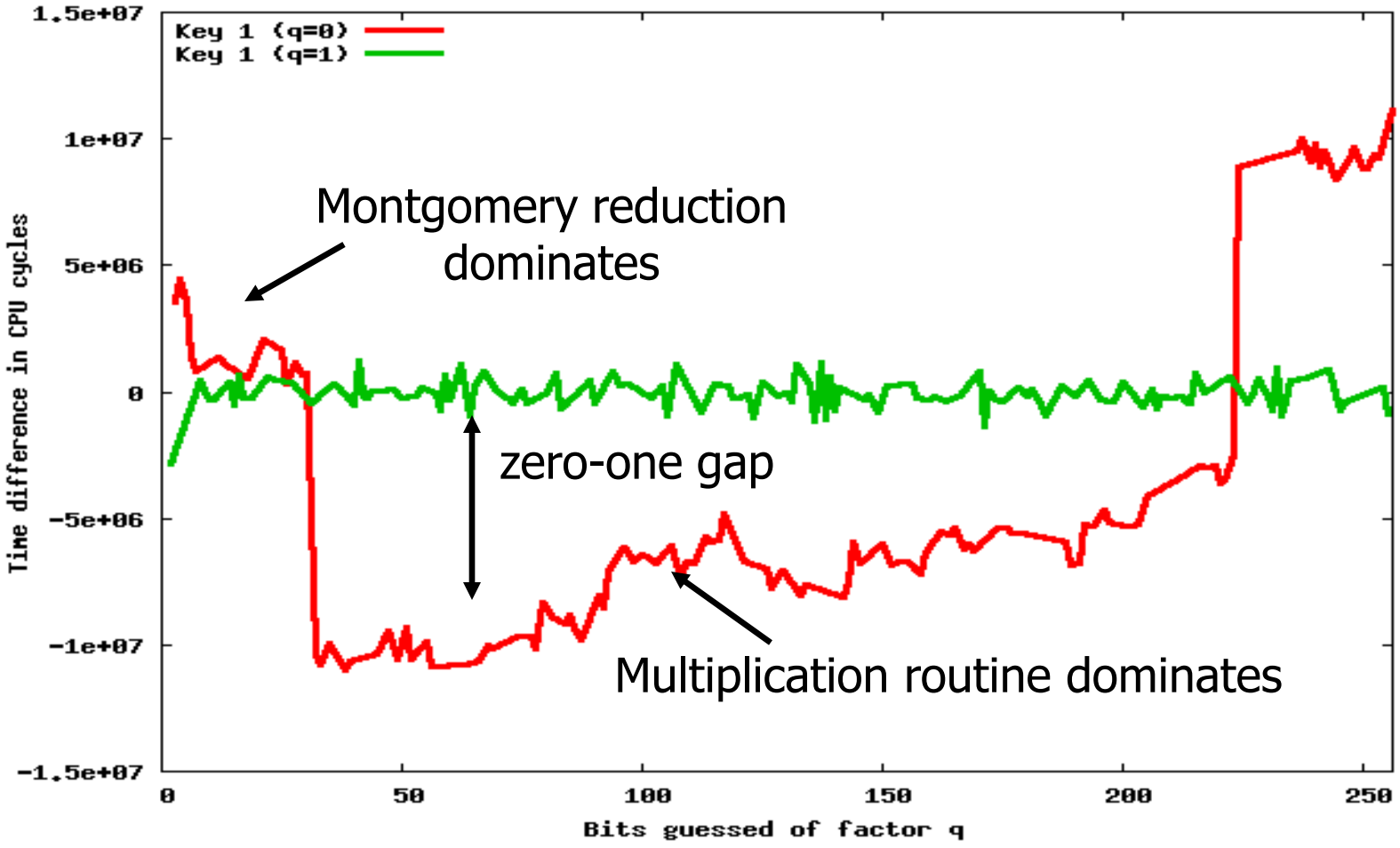
Timing Attack Details

- What is "large" and "small"?
  - Know from attacking previous bits
- Decrypting just g does not work because of sliding windows
  - Decrypt a neighborhood of values near g
  - Will increase difference between large and small values, resulting in larger 0-1 gap
- Attack requires only 2 hours, about 1.4 million queries to recover the private key
  - Only need to recover most significant half bits of q

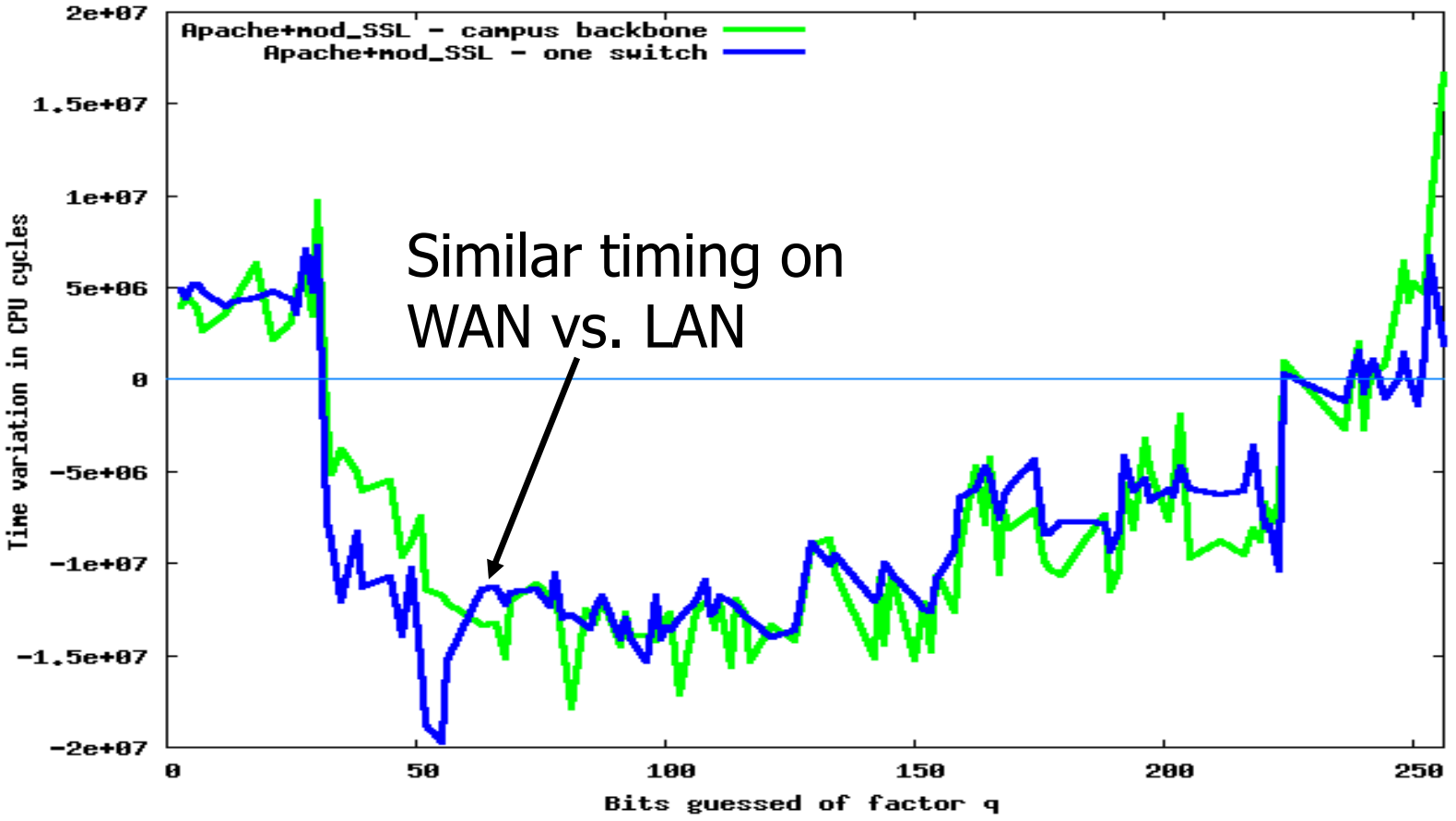
Impact of Neighborhood Size



Extracting RSA Private Key



Works On The Internet



Defenses

- Bad: require statically that all decryptions take the same time
  - For example, always do the extra "dummy" reduction
  - … but what if compiler optimizes it away?
- Bad: dynamically make all decryptions the same or multiples of the same time "quantum"
  - Now all decryptions have to be as slow as the slowest decryption
- Good: Use RSA blinding

RSA Blinding

- Instead of decrypting ciphertext c, decrypt a random ciphertext related to c
  - Compute $x' = c*r^e \bmod n$, r is random
  - Decrypt x' to obtain m'
  - Calculate original plaintext $m = m'/r \bmod n$
- Since r is random, decryption time is random
- 2-10% performance penalty

Blinding Works