

CS 1699

Privacy in the Electronic Society

William Garrison

bill@cs.pitt.edu

6311 Sennott Square

<http://cs.pitt.edu/~bill/1699>

07: Classic side-channel attacks

Continuing: Why isn't crypto enough?

Paul Kocher's timing attack (1996)

- Recover keys by observing **timing variance**
- Works even with seemingly constant-time multiplication
 - Montgomery multiplication

Protections from timing attacks

- Noise
- Extra care to achieve “constant-time”
- Homomorphic **blinding**

Other “classical” side-channel attacks

- Differential power analysis
- Cache timing

First, a basic modular exponentiation algorithm

To compute $y^x \bmod n$:

Let $r = 1$

For $k = 0$ to $b - 1$:

 If bit k of x is 1:

 Let $r = (r \cdot y) \bmod n$

 If not last bit:

 Let $r = r^2 \bmod n$

Return r

$5^{17} \bmod 131$: $y = 5, n = 131$

$x = 17 = 0b10001$

$r = 1$

- $k = 0$, bit $k = 1$

$r = (1 \cdot 5) \bmod 131 = 5$

$r = 5^2 \bmod 131 = 25$

- $k = 1$, bit $k = 0$

$r = 25^2 \bmod 131 = 101$

- $k = 2$, bit $k = 0$

$r = 101^2 \bmod 131 = 114$

- $k = 3$, bit $k = 0$

$r = 114^2 \bmod 131 = 27$

- $k = 4$, bit $k = 1$

$r = (27 * 5) \bmod 131 = 4$

Montgomery multiplications

*Especially good for
repeated multiplications!*

Modular multiplications can be expensive due to divisions

- To compute $a \cdot b \bmod n$, compute $a \cdot b$, then **divide by** n and take remainder

Montgomery form: Replace each number a with $\bar{a} = aR \bmod n$, where R is easy to divide out

- Imagine $R = 100$; divide by shifting as long as trailing 0s
- R must be greater than, and coprime with, n
- Usually $R = 2^k$, where k is the bit length of n
- R is **easy** to multiply in (and divide out via shift, if it divides evenly)

Instead of computing $a \cdot b \bmod n$, compute $\bar{a} \cdot \bar{b} \bmod n = abRR \bmod n$

- Need to eliminate extra R by dividing
- What if R doesn't divide it evenly? Find a value equivalent mod n that it does!
- $abRR + kn \equiv abRR \pmod{n}$ for any k , find a k that results in multiple of R

Even with “constant-time” multiplications, timing information leaks key info

Idea: Want to know the exponent (think RSA, Diffie-Hellman)

- Observe timings for a decryption/signing
- Use statistics to approximate **likelihood** that multiplication step was completed for the first bit
- Predict this bit, use this to determine next bit
- ...

What do I need to know to carry this out?

- Need y, n to determine x in $y^x \bmod n$
- What does this information correspond to in, say, RSA?
- What **type of attack** is this? How could one achieve this?

Some mathematical details of the attack

Let:

- t_i be the time required for multiplication and squaring for bit i
- e be the loop overhead, measurement error, etc.
- Thus, $T = e + \sum_{i=0}^{w-1} t_i$
- Note: These are not constant values, but random variables!

If we can guess x_b (first b bits of x), we can approximate $T_b = \sum_{i=0}^{b-1} t_i$

- Compute $T - T_b = e + \sum_{i=0}^{w-1} t_i - \sum_{i=0}^{b-1} t_i = e + \sum_{i=b}^{w-1} t_i$
- Recall summation rules!

If we can observe the variance of $T - T_b$ for a large sample, we can predict further bits!

- (assuming we are correct about x_b)

Multiple rounds are independent! This means that variance is linear

Assume x_b is correct:

- $V(T - T_b) = V(e + \sum_{i=b}^{w-1} t_i) = V(e) + (w - b)V(t)$

However, if x_b is not correct (say the first error is bit c):

- $V(T - T_b) = V(e) + (w - b + 2c)V(t)$
 - Our variance measurements for bits between c and b are different from reality
 - $2c$ because the variance adds for these bits, rather than cancelling

Thus, a correct guess for the next bit **reduces** $V(T - T_b)$, incorrect guess **increases**!

So, guess the first b bits, then measure to guess the next

That means that this method is error-correcting!

If we get in a position where both bit values **increase** the variance, an earlier bit is probably wrong

How does this help us?

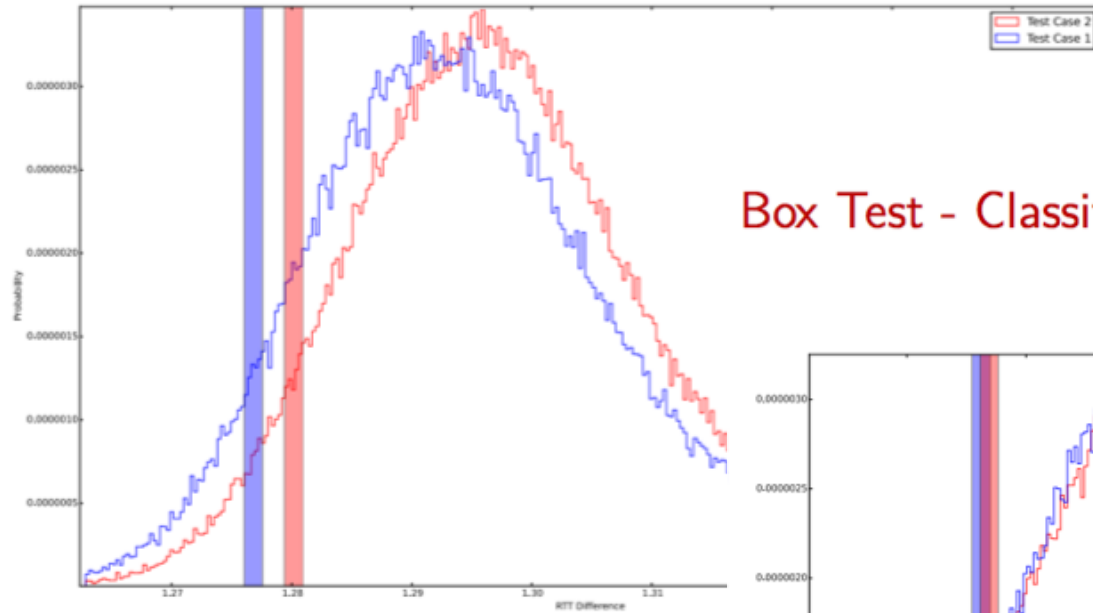
- How do we take advantage to improve the attack?
- Can move on to next bit even with **moderate confidence**, knowing we can backtrack if confidence drops later
- Keep several guesses and levels of confidence, work on whichever is highest likelihood

Is this really practical over the internet?

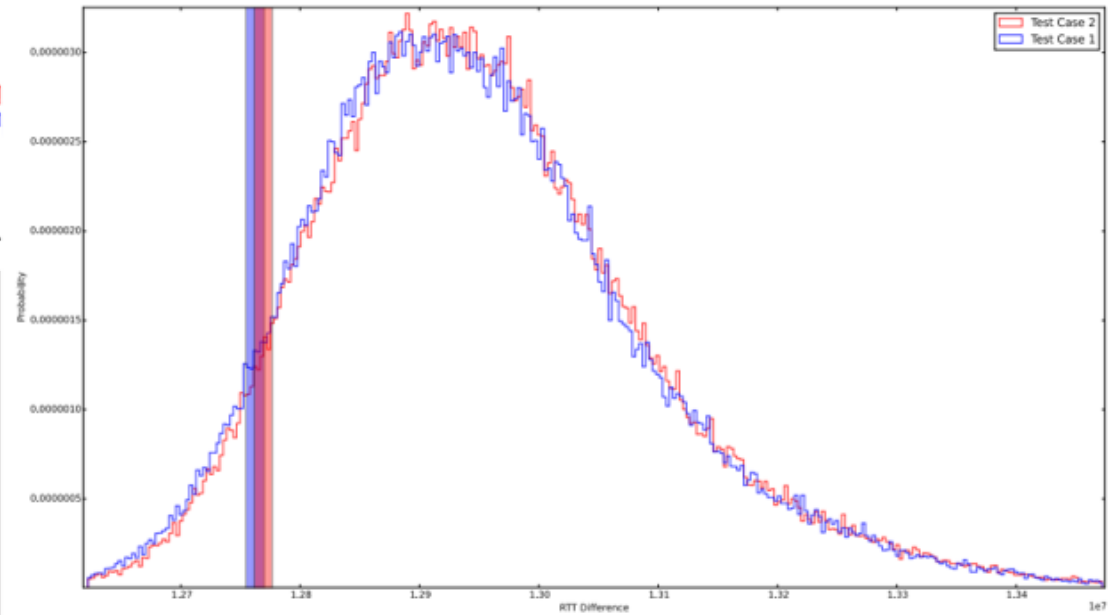
Timothy and Jason Morgan (2015) made progress in achieving this attack in practice

- Main problem: **Noise**
- Network delay noise drowns out signal in simple statistics
- Improvement: Use **TCP timestamps** added as RFC 1323
 - Added for better performance; estimate RTT to know when to resend
 - In timing attack, improves timing of RTT and reduces noise
- Improvement: Use Crosby's **box test**
 - Less susceptible to noise vs. variance
 - “Box in” a range of percentiles, determine if 2 distributions are the same by studying overlap

Box Test - Classified as Different



Box Test - Classified as the Same



So, how can we mitigate such attacks?

We could add a random delay to increase noise

- Downsides? **How much** noise is needed? Is it feasible?

Do better on constant-time operations

- Execute the **union** of statements for all cases, but only save those that are needed
- Downsides? Issues?

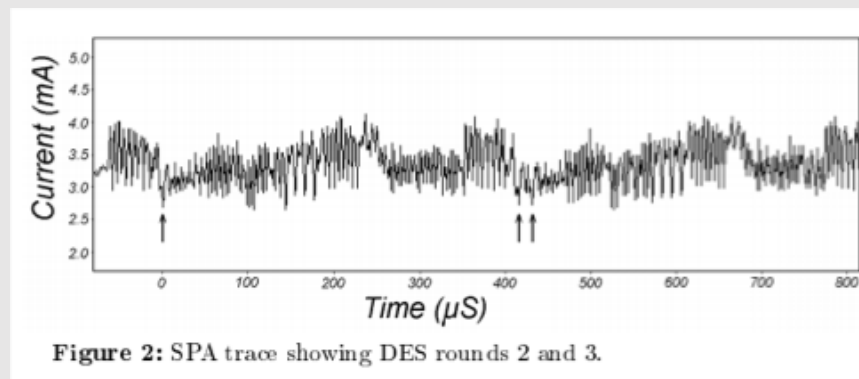
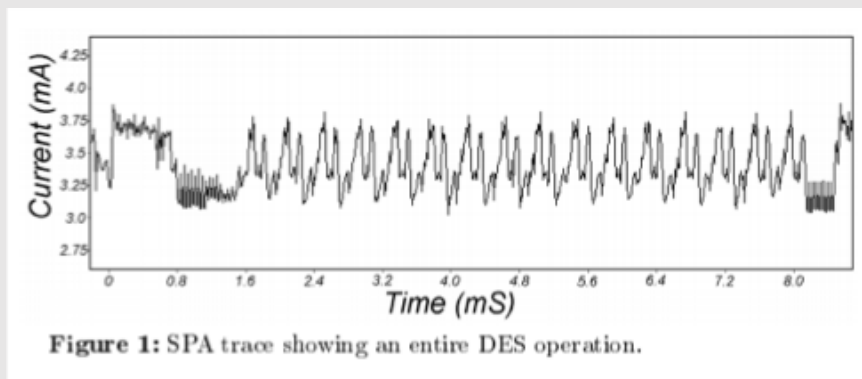
Consider a blinding approach utilizing RSA's multiplicative homomorphism

- Generate pair (u, v) where $u^d = v^{-1} \bmod n$ (i.e., $u^d v = 1 \bmod n$)
- Instead of $c^d \bmod n$, compute $v(uc)^d \bmod n = vu^d c^d \bmod n = c^d \bmod n$
- Timing modexp is really seeing $(uc)^d \bmod n$
- (u, v) should be **secret** and **fresh**; reuse can be revealed in timing
 - Kocher proposed $(u', v') = (u^2 \bmod n, v^2 \bmod n)$ as an update function

Similar attacks on power

Simple Power Analysis (SPA) inspects power graph for spikes corresponding to high-power operations

- Determine key based on when spikes occur



Differential Power Analysis (Kocher et al.) uses statistical analysis on the change in power

- Similar in the abstract to the timing attack we discussed today

Cache-timing attacks

High-level idea: Recently-accessed values are **faster** to access successively

- Recall S-boxes implementing non-linear transformation in AES
- If co-located, cache timing may reveal **which** S-box fields were used
- This, in turn, can leak key information
- See Bernstein 2005

Conclusions

Cryptography (still) isn't enough to protect our privacy alone

- Computers exist in, and interact with, the **real world**
- Timing reveals information about **execution**, and hiding it is hard
- Multiuser systems can still **leak** information on modern OSs
- Protect physical access, colocation with **untrusted** code
- **Don't implement your own cryptography for production code**

Next: Less traditional side-channel attacks