This Is The End

Announcements...

- Estimated grades are out
  - But remember, these are estimates....
  - And also remember, for most Grades Don't Matter (much)
- RRR week
  - Schedule of discussions will be up soon
  - No lecture... BUT special treat!
  - IN PERSON during the Tuesday lecture slot (NOT RECORDED, and please don't record!) I will be discussing my personal Project 2 solution... Including two attacks in the autograder my version will fail!
- Final will be very much like the midterm

The Apple Kool-Aid...

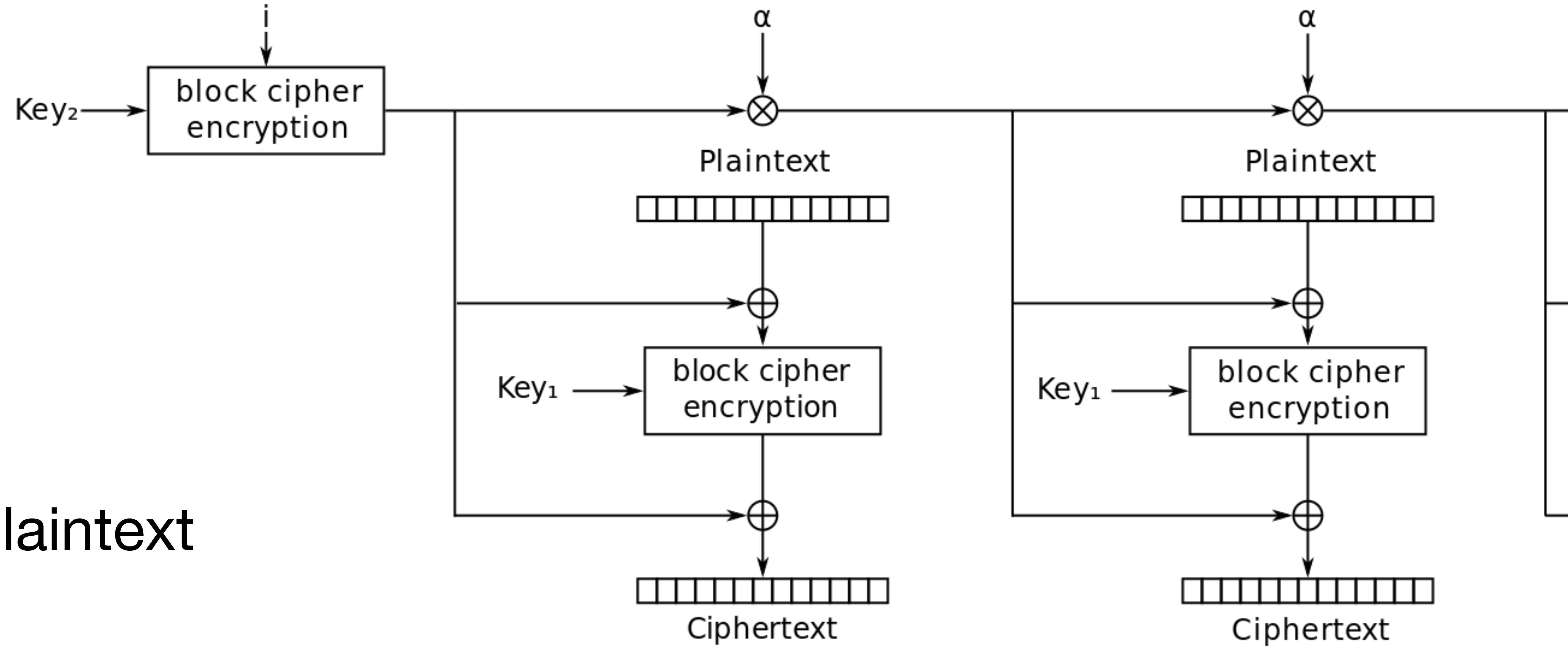
- The iPhone is perhaps the most secure commodity device available...
  - Well, perhaps slightly behind Android, but you can only use Google's Android and Google just wants to spy on you...
  - Not only does it receive patches but since the 5S it gained a dedicated cryptographic coprocessor
- The ***Secure Enclave Processor*** is the trusted base for the phone
  - Even the main operating system isn't fully trusted by the phone!
- A dedicated ARM v7 coprocessor
  - Small amount of memory, a true RNG, cryptographic engine, etc...
  - Important: A collection of ***randomly*** set fuses
    - Should not be able to extract these bits without taking the CPU apart:
Even the Secure Enclave can only use them as keys to the AES engine, not read them directly!
  - But bulk of the memory is shared with the main CPU
- GOOD documentation:
  - The iOS security guide is something you should at least skim....
I find that the design decisions behind how iOS does things make ***great*** final exam questions
- But it isn't perfect: Nation-state actors will pay big $ for exploits
  - So keep it patched

The Roll of the SEP...
Things ***too important*** to allow the OS to handle

- Key management for the encrypted data store
  - The CPU has to ask for access to data!
- Managing the user's passphrase and related information
- User authentication:
  - ***Encrypted*** channel to the fingerprint reader/face recognition camera
- Storing credit cards
  - ApplePay is cheap for merchants ***because it is secure***:
    Designed to have very low probability of fraud!

AES-256-XEX mode

- A ***confidentality-only*** mode developed by Phil Rogaway...
  - Designed for encrypting data within a filesystem block ***i***
    - Known plaintext, when encrypted, can't be replaced to produce known output, only "random" output
  - Within a block: Same cypher text implies different plaintext
  - Between blocks: Same cypher text implies nothing!
  - α is a galios multiplication and is very quick:
    In practice this enables parallel encryption/decryption
- Used by the SEP to encrypt its own memory...
  - Since it has to share main memory with the main processor
- Opens a limited attack surface from the main processor:
  - Main processor can replace 128b blocks with ***random*** corruption



XEX mode encryption

User Passwords...

- Data is encrypted with the user's password
  - When you power on the phone, most data is completely encrypted
- The master key is PBKDF2(password || on-chip-secret)
  - So you need ***both*** to generate the master key
  - Some other data has the key as F(on-chip-secret) for stuff that is always available from boot
- The master keys encrypt a block in the flash that holds all the other keys
  - So if the system can erase this block effectively it can erase the phone by erasing just one block of information
- Apple implemented ***effaceable storage***:
  - After x failures, OS command, whatever...
    Overwrite that master block in the flash securely
  - Destroy the keys == erase everything!

Background: FBI v Apple

- A "terrorist" went on a rampage with a rifle in San Bernardino...
  - Killed several people before being killed in a battle with police
- He left behind a work-owned, passcode-locked iPhone 5 in his other car...
- The FBI ***knew*** there was no valuable information on this phone
  - But never one to refuse a good test case, they tried to compel Apple in court to force Apple to unlock the phone...
- Apple has serious security on the phone
  - Effectively everything is encrypted with PBKDF2(PW||on-chip-secret): >128b of randomly set microscopic fuses
    - Requires that ***any*** brute force attack either be done on the phone or take apart the CPU
  - Multiple timeouts:
    - 5 incorrect passwords -> starts to slow down
    - 10 incorrect passwords -> optional (opt-in) erase-the-phone

What the FBI wanted...

- Apple provides a ***modified*** version of the operating system for the Secure Enclave which...
  - Removes the timeout on all password attempts
  - Enables password attempts through the USB connection
  - Enables an ***on-line*** brute force attack..
    but with a 4-digit PIN and 10 tries/second, you do the math...
- Apple ***cryptographically signs the rogue OS version!***
  - A horrific precedent:
    This is ***requiring*** that Apple both create a malicious version of the OS and sign it
    - If the FBI could compel Apple to do this, the NSA could too...
      It would make it ***impossible*** to trust software updates!

Updating the SEP To Prevent This Possibility...

- The SEP will only accept updates ***signed by Apple***
  - But an updated SEP could exfiltrate the secret to enable an offline attack
- The FBI previously asked for this capability against a non-SEP equipped phone
  - "Hey Apple, cryptographically sign a corrupted version of the OS so that we can brute-force a password"
- How to prevent the FBI from asking again?
- Now, an OS update (either to the base OS and/or the SEP) requires the user to be logged in ***and input the password***
  - "To rekey the lock, you must first unlock the lock"
  - The FBI can only even ***attempt*** to ask before they have possession of the phone since once they have the phone they must also have the passcode
  - So when offered the chance to try again with a "Lone Wolf's" iPhone in the Texas church shooting, they haven't bothered
- At this point, Apple has now gone back and allows auto-updates for the base OS
  - (but probably not the SEP)

The Limits of the SEP... The host O/S

- The SEP can keep the host OS from accessing things it shouldn't...
  - Credit cards stored for ApplePay, your fingerprint, etc...
- But it can't keep the host OS from things it is supposed to access
  - All the user data when the user is logged in...
- So do have to rely on the host OS as part of ***my*** TCB
  - Fortunately it is updated continuously when vulnerabilities are found
    - Apple has responded to the discovery of very targeted zero-days in <30 days
  - And Apple has both good sandboxing of user applications and a history of decent vetting
    - So the random apps are ***not*** in the Trusted Base.

The SEP and Apple Pay

- The SEP is what makes ApplePay possible
  - It handles the authentication to the user with the fingerprint reader/face reader
    - Verifies that it is the user not somebody random
  - It handles the emulation of the credit card
    - A "tokenized" Near Field Communication (NFC) wireless protocol
    - And a tokenized public key protocol for payments through the app
- ***Very hard*** to conduct a fraudulent transaction
  - Designed to enforce user consent at the SEP
- ***Disadvantage***: The fingerprint reader is part of the trust domain
  - Which means you need special permission from Apple to replace the fingerprint reader when replacing a broken screen

I ***love*** ApplePay...

- It is a ***faster*** protocol than the chip-and-signature
  - NFC protocol is designed to do the same operation in less time because the protocol is newer
- It is a ***more secure*** protocol than NFC on the credit card
  - Since it actually enforces user-consent
- It is more ***privacy sensitive*** than standard credit card payments
  - Generates a unique token for each transaction:
    Merchant is not supposed to link your transactions
- Result is its low cost:
  - Very hard to commit fraud -> less cost to transact
- I use it on my watch all the time

Transitive Trust in the Apple Ecosystem...

- The most trusted item is the iPhone SEP
  - Assumed to be rock-solid
  - Fingerprint reader/face reader allows it to be convenient
- The watch trusts the phone
  - The pairing process includes a cryptographic key exchange mediated by close proximity and the camera
  - So Unlock the phone -> Unlock the watch
- My computer trusts my watch
  - Distance-bounded cryptographic protocol
  - So my watch unlocks my computer
- Result? I don't have to keep retyping my password
  - Allows the use of ***strong passwords everywhere*** without driving myself crazy!

Credit Card Fraud

- Under US law we have very good protections against fraud
  - Theoretical $50 limit if we catch it quickly
  - $0 limit in practice
- So cost of credit card fraud for me is the cost of recovery from fraud
  - Because fraud ***will happen***:
  - The mag stripe is all that is needed to duplicate a swipe-card
    - And you can still use swipe-only at gas pumps and other such locations
  - The numbers front and back is all that is needed for card-not-present fraud
    - And how many systems
- What are the recovery costs?
  - Being without the card for a couple of days...
    - Have a second back-up card
  - Having to change all my autopay items...
    - Grrrr....

But What About "Debit" Cards?

- Theoretically the fraud protection is the same...
- But two caveats...
  - It is easier to not pay your credit card company than to claw money back from your bank...
  - Until the situation is resolved:
    - Credit card? It is the credit card company's money that is missing
    - Debit card? It is ***your*** money that is missing
- Result is debit card fraud is more transient disruptions...

So Two Different Policies...

- Credit card: Hakunna Matata!
  - I use it without reservation, just with a spare in case something happens
  - Probably 2-3 compromise events have happened, and its annoying but ah well
    - The most interesting was $1 to Tsunami relief in 2004...
      was a way for the attacker to test that the stolen card was valid
- Debit card: Paranoia-city...
  - It is an ATM-ONLY card (no Visa/Mastercard logo!)
  - It is used ONLY in ATMs belonging to my bank
    - Reduce the risk of "skimmers": rogue ATMs that record cards and keystrokes

Putting Everything Together In the Real World: The "Sad DNS" Attack...

- Over a decade after the Kaminski attacks, DNS cache poisoning is back in the news
- Reminder: Kaminski strategy...
  - You send glue records to actually poison the target:
    So to poison **`www.google.com`**, you create a query for **`a.google.com`**...
    And in the additional include **`www.google.com A 66.66.66.66`**
  - Still have to guess TXID (2^{16} work factor), but can keep trying!
- Defense was randomize the UDP source port as well...
  - So attacker has to guess the port and TXID at the same time (so 2^{32} work give-or-take)

Work by Keyu Man et al.. saddns.net (UC Riverside & Tsinghua University)

- Observation #1, can we detect what UDP port(s) are in use for a particular query?
  - If so, it turns the problem from expected 2^{32} work to 2^{16} + 2^{16} work!
  - You search for the open port, and if you get lucky, do the random TXIDs...
- Observation #2, can we cause the DNS authority for a domain to ***not respond?***
  - If so, enables us to have a lot more time for an attack
  - Which can make it far easier to be successful
- Answer to both is ***yes!***

Answer to 1: Just Ask the DNS Resolver!

- By default you get a response if there is no open UDP port
  - "ICMP port unreachable"
- And UDP ports are not host specific by default...
  - So if you call **`sendto()`** and then **`recvfrom()`**...
    you won't send an ICMP back for that port
  - Behavior is not the same for **`connect()`** semantics:
    Connect will only not send back an ICMP if the UDP packet is from the remote IP
- So just scan all 2^{16} ports to see if you get a response!
- But there are gotchas...
  - ICMP packet sending has both a per-IP and global rate limit

So Enter Side Channels....

- Spoof a bunch of packets that will ***just*** trigger the global rate limit
  - Then send a packet from your IP to a port you ***know*** will trigger an ICMP response
- If you get a response...
  - One of the ports you checked was open!
  - So divide and conquer
- If you don't... Wait the short 20ms timeout and go onto the next block of ports to check
- Oh, and if they use **`connect()`** for UDP...
  - You only don't get an ICMP back if you are the IP that was connected to...
    So just spoof the real server with the side channel check!

And Now To Buy Time... Another Rate Limit...

- DNS servers can be used for reflected DOS attacks
  - Spoof the IP address of the target and send a packet to the DNS server
  - DNS server then replies...
    Making the attack look like its coming from the DNS server
  - And since DNS replies are bigger, this is an amplifier for DOS attacks
- So DNS authority servers have their own rate limit
  - Too many requests from a single IP and they will start ignoring some request
- So use ***that*** to buy time...
  - Send just enough requests spoofing the target resolver's IP address for nonsense requests
  - Target resolver ignores the replies (after all, they were never made)
  - But the DNS authority server will now ignore the target resolver's DNS request!

Solution #1: DNSSEC

- If the resolver (or better yet client) validates DNSSEC...
  - Now it doesn't matter!
- Fortunately DNSSEC serving is getting easier
  - Most people are using a few outsourced DNS services
  - So they can easily add in DNSSEC if they aren't already
  - ***Any*** managed DNS service should use DNSSEC these days

Solution #2:
Detection & Response

- Still relies on Kaminsky-style glue records for poisoning
  - Otherwise you can only race once on failure until the record's TTL expires
- This is ***VERY NOISY***
  - Hundreds or thousands of non-matching responses
  - This is even noisier than standard Kaminsky:
    Lots of bogus replies from the real server to suppress the legitimate reply
- So detect and respond
  - Don't query once, query multiple times and accept majority
  - Don't promote glue into the cache
  - Or just don't resolve the targeted name(s)
  - Nobody does this however

And Now:
Ask Me Anything!