Applied Crypto: Passwords & Signal & Tor

USE SIGNAL!  ¿Use TOR?
Administrivia: Exam Logistics

- [https://cs161.org/exam](https://cs161.org/exam)
  - READ IT: This is just the tl;dr summary
  - And do Homework 3 now...
    At least the part about the exam logistics!
  - Scope is everything up to and including this lecture

- Basic concept
  - You have arbitrary *hand-written* paper notes
    - You can compose them on a tablet with a stylus interface but you need to print them out
  - You do the exam on your computer from a pre-distributed encrypted PDF
  - Proctoring is through your phone/second device over Zoom
  - Possible of a "Trust but verify" quick oral quiz afterwards
    - Explain how to solve a variant of a question you successfully solved on the exam
Why this structure?

• We need to do what we can to ensure academic honesty
  • We also have hidden techniques we are using as well
  • If I told all of you "The exam is unproctored and we have no mechanism to detect, respond, or validate that you don't cheat"...
  • I would be **personally insulted and a total failure if 100% of you DIDN'T CHEAT!**
    • I want my students to be rational, and under those conditions it would be irrational not to cheat

• At the same time, there is a non-linear relationship between effectiveness and burden
  • I doubt the aggressive CS162-style starring on reddit 17 page protocol is significantly better at dissuading dishonesty
Concerns about student privacy

- You will be notified in advance of which TA will be proctoring you
  - For any reason you can switch proctors if you reach out in advance
- You can join using your SID rather than your name
  - Plus zoom on phone sucks for looking at other people's rooms
- If you are not comfortable with this setup, reach out for arranging off-line recording
- If you temporarily lose connectivity, don't worry
  - Focus on the exam, not your Internet connection
- If you need to get up and go to the bathroom, stretch, etc... Go ahead
Reminder: Cryptographic Hashes...

- We love ourselves some cryptographic hashes
  - SHA_256, SHA_384, SHA3_256, SHA3_384

- Reminder on the properties:
  - Irreversible:
    Given $H(X)$, it is infeasible to find $X$ short of simply trying all possibilities
  - First preimage resistant:
    Given $H(X)$, it is infeasible to find any $X'$ such that $H(X) = H(X')$
  - Second preimage resistant:
    It is infeasible to find $X$ and $Y$ such that $X \neq Y$ and $H(X) = H(Y)$
A Couple Other Hash Properties...

- They accept arbitrarily large inputs
- They "look" random
  - Change a single bit on the input and each output bit has a 50% chance of flipping
  - And until you change the input, you can't predict which output bits are going to change
- The ones we talked about are fast
  - Can operate at many many MB/s:
    Faster at processing data than block ciphers
A Hash Problem: Proof of work...

- Alice wants Bob to waste a bunch of CPU resources
- But wants to quickly check that Bob wasted that much CPU
- Alice -> Bob: "Here is a message $M$ and a factor $x$"
  - Make sure $M$ has a nonce in it
- Now Bob needs to provide $M'$ such that it starts with $M$ and $H(M')$ starts with $x$ zero bits
- Alice computes $H(M')$ and verifies that it starts with $x$ zero bits
  - Alice now knows that Bob is expected to have had to create $2^x$ separate M's and hash them until he found one that matched
What this provides

- You can use it in a protocol where the user has to waste something...
  - EG, proposals for sending mail as a way of reducing spam
    - It wouldn’t: Bad guys can get lots of CPU resources

- Have other options too

- CAPTCHAs:
  - Those "prove your human" web puzzles:
    - It is a proof you wasted a few seconds of a human's time
      - (Or that you paid $.01 to waste a few seconds of a human's time)

- Proof of *wait*
  - Alice has a secret key $k$
  - Alice to Bob sends "Don't contact me until time $T$, here is HMAC($k,T$)"
  - When Bob gets back, he says "$T$, HMAC($k,T$)"
  - Alice then verifies $T$ is in the past and HMAC($k,T$)
Passwords

• The password problem:
  • User Alice authenticates herself with a password $P$
  • How does the site verify later that Alice knows $P$?

• Classic:
  • Just store $\{\text{Alice, } P\}$ in a file...
  • But what happens when the site is hacked?
    • The attacker now knows Alice's password!

• Enter "Password Hashing"
Password Hashing

• Instead of storing \{Alice, P\}...
  • Store \{Alice, H(P)\}

• To verify Alice, when she presents P
  • Compute \(H(P)\) and compare it with the stored value

• Problem: Brute Force tables...
  • Most people chose bad passwords...
    And these passwords are known
  • Bad guy has a huge file...
    • \(H(P1), P1\)
    • \(H(P2), P2\)
    • \(H(P3), P3\)...
  • Ways to make this more efficient ("Rainbow Tables")
A Sprinkle of Salt...

• Instead of storing \{Alice, H(P)\}, also have a user-specific string, the "Salt"

• Now store \{Alice, Salt, H(P||Salt)\}

• The salt ideally should be both long and random, but it isn't considered "secret": rather it is a nonce

• As long as the salt is unique...

• An attacker who captures the password file has to brute force Alice's password on its own

• Its still an "off-line attack" (Attacker can do all the computation he wants) but...

• At least the attacker can't precompute possible solutions
Slower Hashes...

- Most cryptographic hashes are designed to be fast
  - After all, that is the point: they should not only turn $H(🐮)$ to hamburger... they need to do it quickly

- But for password hashes, we want it to be slow!
  - It's OK if it takes a good fraction of a second to check a password
    - Since you only need to do it once for each legitimate usage of that password
    - But the attacker needs to do it for each password he wants to try

- Slower hashes don't change the asymptotic difficulty of password cracking but can have huge practical impact
  - Slow rate by a factor of 10,000 or more!
PBKDF2

• "Password Based Key Derivation Function 2"
  • Designed to produce a long "random" bitstream derived from the password
  • Used for both a password hash and to generate keys derived from a user's password

PBKDF(PRF, P, S, c, len):
  • PRF == Pseudo Random Function (e.g. HMAC-SHA256)
  • P == Password
  • S == Salt
  • c == Iteration count
  • len == Number of bits/bytes requested
  • DK == Derived Key

PBKDF(PRF, P, S, c, len) {
  DK = ""
  for i = 1,range(len/blocksize)+1{
    DK = DK || F(PRF,P,S,c,i)
  }
  return DK[0:len]
}

F(PRF,P,S,c,i) {
  UR = U = PRF(P, S||INT_32(i))
  for j = 2; j <= c; ++j {
    U = PRF(P, U)
    UR = UR ^ U
  }
  return UR
}
Comments on PBKDF2

• Allows you to get effectively an arbitrary long string from a password
  • Assuming the user's password is strong/high entropy

• Very good for getting a bunch of symmetric keys from a single password
  • You can also use this to seed a PRNG for generating a "random" public/private key pair

• Designed to be slow in computation...
  • But it does not require a lot of memory:
    Other functions are also expensive in memory as well, e.g. scrypt & argon2
Passwords...

- If an attacker can do an **offline** attack, your password must be **really good**
  - Attacker simply tries a huge number of passwords in parallel using a GPU-based computer: buy a bunch of used Nvidia 2080 supers from all those upgrading to 3080s
  - So you need a **high entropy** password:
    - Even xkcd-style is only 10b/word with a 1000 word dictionary, so need a 7 or more **random word** passphrase to resist a determined attacker
- Life is far better is if the attacker can only do **online** attacks:
  - Query the device and see if it works
  - Now limited to a few tries per second and no parallelism!
... and iPhones

- **Apple's security philosophy:**
  - In your hands, the phone should be everything
  - In anybody else's, it should (ideally) be an inert "brick"

- **Apple uses a small co-processor in the phone to handle the cryptography**
  - The "Secure Enclave"

- **The rest of the phone is untrusted**
  - Notably the memory: *All* data must be encrypted:
    The CPU requests that the Secure Enclave unencrypt data and some data (e.g., your credit card for ApplePay) is only readable by the Secure Enclave

- **They also have an ability to effectively erase a small piece of memory**
  - "Effaceable Storage": this takes a good amount of EE trickery
Crypto and the iPhone Filesystem

- A lot of keys encrypted by keys...
  - But there is a random master key, $k_{\text{phone}}$, that is the root of all the other keys
- Need to store $k_{\text{phone}}$ encrypted by the user’s password in the flash memory
  - $\text{PBKDF2}(P, ...) = k_{\text{user}}$
- But how to prevent an off-line brute-force attack?
  - Also have a 256b random secret burned into the Secure Enclave that you can use for encryption
    - Need to take apart the chip to get this!
  - Even the secure enclave can’t read this secret, only use this secret as a key for hardware cryptographic engines
- Now the user key is not just a function of $P$, but $E(K_{\text{secret}}, P)$
  - Without the secret, can not do an offline attack
- All online attacks have to go through the secure enclave
  - After 5 tries, starts to slow down
  - After 10 tries, can (optionally) nuke $k_{\text{phone}}$!
    - Erase just that part of memory -> effectively erases the entire phone!
    - Even compromising the secure enclave limits guessing to 10 per second!
Backups...

• Of course there is a necessary weakness:
  • Backing up the phone copies all the data off in a form not encrypted using the in-chip secret
    • After all, you need to be able to recover it onto a new phone!

• So someone who can get your phone...
  And can somehow managed to have it unlocked
  • Thief, abusive boyfriend, cop...
    • Hold it up to your face (iPhone X) or Fingerprint (5s or beyond)
    • And then sync it with a new computer

• Change of policy for iOS-11:
  • Now you also need to put in the passcode to trust a new computer:
    Can't create a backup without knowing the passcode
Signal and Tor

- Signal is a messenger protocol and implementation
  - Signal (the company) is a 501(c)3 nonprofit
  - The protocol is also used by WhatsApp, Facebook Messenger, etc...

- Tor is an anonymity tool
  - Designed to provide anonymous but real-time network connectivity in the face of an aggressive but local adversary

- Common (bad) information security advice is "Use Signal, Use Tor"
  - In reality, Signal is a great protocol, but some security compromises are annoying in the implementation, so for most, WhatsApp is about as good
  - While Tor is often not just a placebo but *poison*!
End-To-End Messengers

• We love end to end cryptographic protocols...
  • After all, we just saw why we might want them

• We love forward secrecy...
  • After all, we want things to stay secret even if our keys are compromised

• Forward secrecy is "easy" for online protocols
  • Just make sure to do a DHE/ECDHE key exchange, and throw away the session key when done

• Forward secrecy is much more annoying for an offline protocol
  • Alice wants to share data with Bob, but Bob is not online
    • Like in project 2...
    • Or any messenger system!
Signal Requirements For Key Agreement

• Three parties: Alice, Bob, and a messenger server
  • The messenger server is like the file store in project 2, an untrusted entity
  • A separate mechanism is used to provide key transparency

• Bob is offline:
  • He has prearranged data stored on the messenger server

• Alice and Bob want to create an ephemeral (DH) key...
  • To use for then encrypting messages

• They need mutual authentication
  • Assuming Alice and Bob have the correct public keys, only Alice and Bob could have agreed on a key

• They also need deniability
  • Alice or Bob can't create a record proving the other side participated in creating the key:
    So no "Alice just signs her DH..." design
Extended Triple Diffie-Hellman

• Key idea:
  • Lets use multiple Diffie-Hellman exchanges combined into one
    • Some to perform mutual authentication
    • Some to generate an ephemeral key
    • Shove them ALL into a hash-based key derivation function

• They use elliptic curves, but the design would be the same for conventional DH, so we will use the former
  • We will use $DH(A,B)$ as $DH(g^a,g^b)$ where we know $a$ but not $b$. (So $A$ is our private value, $B$ is someone else's public value)
  • Also have $Sign(K,M)$ for signing and $KDF(KM)$ which derives a bunch of session keys for a hash-based key derivation function (e.g. $PBKDF2$ with only a couple iterations)
Lots of Keys!

- All keys have both a public & private component
  - Private components always stay with Alice and Bob
  - Anything broadcast is always the public component

- Alice:
  - $IK_A$: Alice's identity key: for both DH and signatures
  - $EK_A$: Alice's ephemeral key: Created randomly just to talk to Bob.

- Bob:
  - $IK_B$: Bob's identity key, long lived
  - $SPK_B$: Bob's signed rekey, rotates ~weekly/monthly
    - Has corresponding signature $\text{Sign}(IK_b, SPK_b)$
  - $OPK_B$: Bob's one time use keys (One Time Prekey)
    - Can run out, but designed to increase security when available
Before We Start:
Bob to Server, Server to Alice

- Bob uploads:
  - $IK_B$, $SPK_B$, $\text{Sign}(IK_B, SPK_B)$, $\{OPK_B^1, OPK_B^2, OPK_B^3 \ldots\}$
- Now when Alice wants to talk to Bob...
- Gets from the server:
  - $IK_B$, $SPK_B$, $\text{Sign}(IK_B, SPK_B)$, $OPK_B^?$
  - Told which $OPK$ it is or "There are no $OPK$s left"
    - $OPK$s are designed to prevent replay attacks:
      Bob will \textit{never} allow any particular $OPK$ to be used twice
- This is now the input into Alice's DH calculations
Alice now does a lot of DH...

- **DH1 = DK(IK_A, SPK_B)**
  - Acts as authentication for Alice when Bob does the same
- **DH2 = DK(EK_A, IK_B)**
  - Forces Bob to do mutual authentication
- **DH3 = DK(EK_A, SPK_B)**
  - Adds in ephemeral $EK_A$ to short lived $SPK_B$
- **DH4 = DK(EK_A, OPK_B)**
  - Adds in one-time used $OPK_B$, if available
- **SK = HKDF(DH1 || DH2 || DH3 || DH4)**
  - Skip DH4 if no one time pre-keys are available
  - Now discard the private part of EKA and the intermediate DH calculations
Now Alice Sends To Bob

- $IK_A, EK_A$, which $OPK$ used (if any), and $E(SK, M, IK_A \parallel IK_B)$

- Using an AEAD encryption mode: 
  **Authenticated Encryption with Additional Data** modes allow additional data to be protected by the MAC but sent in the clear:
  In this case $IK_A$ and $IK_B$

- Bob can do the same DH calculations to generate SK
  - Since Bob knows the private keys corresponding to the public values Alice used
  - If it fails to verify the AEAD data abort
Key Transparency

• For now, Alice and Bob are trusting the server to report $IK_A$ and $IK_B$ correctly
  • If the server lies, 🤔

• Fortunately there is an answer:
  If Alice and Bob are ever together:
    • One person's phone displays $H(IK_A \parallel IK_B)$ as a QR Code
    • Other person's phone verifies that it is the same

• Plus the voice channel...
  • Display "Two Words" on screen: $F(H(IK_A \parallel IK_B \parallel SK))$
  • Assumption is a MitM attacker can't fake a voice conversation quickly enough, so if each person says one of the words...
Considerations

• Authentication requires the out-of-channel methods
  • Otherwise no guarantees

• Replay attacks
  • Only if no OPK is available: Can be potentially bad

• Deniability
  • No cryptographic proofs available as to the sender/receiver!
And Then Ratchets...

- A "ratchet" is a one-way function for message keys
  - \( Ratchet(K_i) \rightarrow K_{i+1}, MK_i \)
  - But can't take \( K_{i+1} \) and \( MK_i \) to find \( K_i \)

- A symmetric key ratchet is easy
  - We've seen these already:
    Any secure PRNG with rollback resistance is a ratchet
  - Can do it slightly more efficiently with HMAC:
    \( \text{HMAC}(K_i, 0x01) \rightarrow MK_i \)
    \( \text{HMAC}(K_i, 0x02) \rightarrow K_{i+1} \)

- It's OK to keep around the intermediate session keys
  - Thanks to HMAC we can't go backwards with them anyway:
    Needed for out of order messages
Signal adds in DH ratchets too...

- So for a few messages in a chain you use a symmetric key ratchet...
  - You gain forward secrecy by discarding the old internal state
- But occasionally you rekey with an additional DH
  - Used to add into the ratchet internal state: update $K_i$ to $H(K_{i-1} \parallel DH)$
- Acts to reset everything with even more randomness
  - So even if you compromise Bob's device at time $T$ and steal all the keys...
  - You can't decrypt old messages that aren't on Bob's device: can't run the symmetric ratchet backwards
  - You can't decrypt subsequent messages once Bob & Alice use a DH ratchet
The Protocol is Great...

BUT!

• The app itself does some ehh thing in the usability/security tradeoff...
  
  • *No mechanism to back-up messages!*  
    If your phone is toast, your messages are gone!
  
  • *No mechanism to migrate to a new phone!*  
    If you upgrade to a new phone, your messages are gone!
  
  • *Auto-notifies all those where you are in their contacts that they join*

• This is where WhatsApp has a huge competitive advantage
  
  • They allow backup of messages, message migration etc...
Tor: The Onion Router
Anonymous Websurfing

- Tor actually encompasses many different components
- The Tor network:
  - Provides a means for anonymous Internet connections with low(ish) latency by relaying connections through multiple Onion Router systems
- The Tor Browser bundle:
  - A copy of FireFox extended release with privacy optimizations, configured to only use the Tor network
- Tor Hidden Services:
  - Services only reachable though the Tor network
- Tor bridges with pluggable transports:
  - Systems to reach the Tor network using encapsulation to evade censorship
- Tor provides three separate capabilities in one package:
  - Client anonymity, censorship resistance, server anonymity
The Tor Threat Model: Anonymity of content against local adversaries

- The goal is to enable users to connect to other systems “anonymously” but with low latency
- The remote system should have no way of knowing the IP address originating traffic
- The local network should have no way of knowing the remote IP address the local user is contacting

Important what is excluded:
- The global adversary
- Tor does not even attempt to counter someone who can see all network traffic: It is probably impossible to do so and be low latency & efficient
The High Level Approach: Onion Routing

- The Tor network consists of thousands of independent Tor nodes, or “Onion Routers”
  - Each node has a distinct public key and communicates with other nodes over TLS connections
- A Tor circuit encrypts the data in a series of layers
  - Each hop away from the client removes a layer of encryption
  - Each hop towards the client adds a layer of encryption
- During circuit establishment, the client establishes a session key with the first hop…
  - And then with the second hop through the first hop
- The client has a *global* view of the Tor Network:
  The directory servers provide a list of all Tor relays and *their public keys*
Tor Routing
In Action
Tor Routing
In Action
Creating the Circuit Layers…

• The client starts out by using an authenticated DHE key exchange with the first node…
  • OR1 creates $g^a$, signs it with its private key, sends $g^a, \text{Sign}(K_{priv\_or1}, g^a)$ to client
  Client creates $g^b$, sends it to OR1
  Client does $\text{Verify}(K_{pub\_or1}, g^a)$
• Creating a session key $K_{OR1}$ as $H(g^{ab})$
  • This first hop is commonly referred to as the “guard node”

• It then tells OR1 to extend this circuit to OR2
  • Through that, creating a session key for the client to talk to OR2 that OR1 \textit{does not know}
  • And OR2 doesn’t know what the client is, just that it is somebody talking to OR1 requesting to extend the connection...

• It then tells OR2 to extend to OR3…
  • And OR1 won’t know where the client is extending the circuit to, only OR2 will
Unwrapping the Onion

• Now the client sends some data…
  • $E(K_{or1}, E(K_{or2}, E(K_{or3}, \text{Data})))$

• OR1 decrypts it and passes on to OR2
  • $E(K_{or2}, E(K_{or3}, \text{Data}))$

• OR2 then passes it on…

• Generally go through at least 3 hops…
  • Why 3? So that OR1 can’t call up OR2 and link everything trivially

• Messages are a fixed-sized payload
The Tor Browser…

- Surfing “anonymously” doesn’t simply depend on hiding your connection…
- But also configuring the browser to make sure it resists tracking
  - No persistent cookies or other data stores
  - *No deviations from other people* running the same browser
- Anonymity *only works in a crowd*…
  - So it really tries to make it all the same
- But by default it makes it easy to say “this person is using Tor”
But You Are Relying On Honest Exit Nodes...

- The exit node, where your traffic goes to the general Internet, is a man-in-the-middle...
  - Who can see and modify all non-encrypted traffic
  - The exit node also does the DNS lookups
  - Exit nodes have not always been honest...
Anonymity Invites Abuse… (Stolen from Penny Arcade)

Unreal Tournament 2004 lends incontrovertible proof to John Gabriel’s Greater Internet Fuckwad Theory.

Normal Person + Anonymity + Audience = Total Fuckwad
This Makes Using Tor Browser Painful…
And Also Makes Running Exit Nodes Painful…

• If you want to receive abuse complaints…
  • Run a Tor Exit Node

• Assuming your ISP even allows it…
  • Since they don’t like complaints either

• Serves as a large limit on Tor in practice:
  • Internal bandwidth is plentiful, but exit node bandwidth is restricted

• Know a colleague who ran an exit node for research…
  • And got a visit from the FBI!
One Example of Abuse: The Harvard Bomb Threat…

- On December 16th, 2013, a Harvard student didn’t want to take his final in “Politics of American Education”…
  - So he emailed a bomb threat using Guerrilla Mail
  - But he was “smart” and used Tor and Tor Browser to access Guerrilla Mail
- Proved easy to track
  - “Hmm, this bomb threat was sent through Tor…”
  - “So who was using Tor on the Harvard campus…” (look in Netflow logs..)
  - “So who is this person…” (look in authentication logs)
  - “Hey FBI agent, wanna go knock on this guy’s door?!”
- There is no magic Operational Security (OPSEC) sauce…
  - And again, anonymity only works if there is a crowd
Censorship Resistance: Pluggable Transports

- Tor is really used by separate communities
  - Anonymity types who want anonymity in their communication
  - Censorship-resistant types who want to communicate despite government action
    - The price for "free" censorship evasion is that your traffic acts to hide other anonymous users

- Vanilla Tor fails the latter completely

- So there is a framework to deploy bridges that encapsulate Tor over some other protocol
  - So if you are in a hostile network...
  - Lots of these, e.g. OBS3 (Obfuscating Protocol 3), OBS4, Meek...
  - But its an arm's race