CS161 Notes - Midterm 2 Material

Paul Legler, Nathan Pucheril

July 29, 2017

Contents

Merkle Tree

1 Bitcoin

- Best policy: ever spend your bitcoin
- A currency where the best policy is to never spend
- If you buy bitcoin you have to wait a few days before you can tell your bank it was a fraud and get your money back
- Although the transaction itself is cheap, waiting time of ten minutes has real world cost
- Bitcoin has no tie to dollars so it fluctuates up and down
- Every actual bitcoin transaction: turning dollars to ditcoing, doing bitcoin, turning to back to dollars
- Using bitcoin is a political statement and is censorship resistant (drugs)
- Silk Road was a TOR hidden service marketplace - US centric - selling drugs and accepted bitcoin - Mandatory feedback and escrow system, optional currency hedge for sellers
  - The Armory: you could buy guns too - amazingly illegal for everyone - black market prices - 95 percent of people just buy guns legally
- Ransomeware: you infect a system, encrypt files, encrypt master key with extortionist's public key, tell them to pay money or they won't see there file again - hard to get paid
- Can't store Bitcoin on an Internet connected device - or you will get robbed - store with a private key offline
- The ECDSA signature scheme has a nonce (k-value) like r in El Gamal. If you sign two different messages with the same k, it becomes trivial to find the private key
  - Green Dot MoneyPak: service that allows you to ransfer money to reload prepaid credit cards for 6 dollars - this used to be preferred by ransomware but not Bitcoin
  - Bitcoin uses a lot of energy -more than UC Berkeley
  - Bitcoin is pseudonymous - every wallet is a distinct pseudonym and every transaction is public
  - Ethereum: "Smart" Contracts in a JavaScript-like Language
    - Executes a small virtual machine - Paymaent to a destination can invoke the destination's program

        - In paying someone else, it invokes code outside itself
        - At the same time, the cryptocurrency community tends to belive code is law - the basis of some very interesting attacks:
            - Denial of Service (DOS): Find code that is cheap in terms of ”gas” but expensive in practice - like dis I/O - spend a bunch of transactions to slow down system and stop the network
            - Distibuted Autonomous Organazation: like a mutual fund whose investments are the consensus of the participants - including the ability to split off and perform other actions - it tended to be a bit of ”natural ponzi” scheme right from the start - nearly 10 percent of all Ethereum was invested in the DAO
              Bug: Attacker could propose a split that he gets all his money - split process would first transfer money the reduce the balance - in transferring the destination is simply calling another function, one being written by the attacker - the attacker can claim he followed the contract (the code) - classic TOCTOU
              This caused the Ethereum community to split in two - one group said ”revise history” and simply have the miners ignore the DAO theft - the other group kept the old chain going because they didn't have their money stolen from them

- Anything electronic must have an undo - detection and mitigation
- Bitcoin has a limited amount of programmibility - you can pay two an address - inputs are actually small scripts
- Allowed to set up a structure where in order for some transaction 2 or 3 signuatures must be present

1.1 How to Make Money in Bitcoin in 10 Easy Steps

- Move to Sochi
- Break into blockchain.info and other web-wallet services
- Download saved web wallets for offline cracking
- Modify wallet service javascript to leak passwords
- Be patient and wait
- When discovered, steal all the Bitcoin
- Blame the victims
- Write malcode to look for Bitcoin wallets
- Crack away and rob everyone
- Enjoy life

2 Key Management

Problem with sending a public key: MITM can change to be his own public key and decry pt any message.

2.1 PKI - Public Key infrastructure

Helps manage public keys and certificates

2.1.1 Trusted Directory

- **Trusted Directory** contains mapping of User to User Public key that cant be interfered with. Trusted Directory has a secret key and public key PK_{TD} and trusted directory is accessible by anyone.
- Trusted Directory signs transmission of data so receiving users can check if there was a Man in the Middle Attack. $Sig[SK_{TD}, PK_B]$. However there is still an issues: MITM can get give Alice his/her own public key and pretend to be bob by passing on Trusted Directory Data to Alice (Alice thinks this is safe because it was 'signed' by TD, however MITM could have sent alice his). To get around this, have TD send the following 'BOBS PUBLIC KEY IS PK_B'. Thus the TD includes data on whose key they are signing and alice can confirm whose key they receive.
- MITM requests Eve's public key, and TD answers with $Sig[SK_{TD}, PK_M]$. Then Eve can send this back to Alice instead of $Sig[SK_{TD}, PK_B]$ Instead, send $Sig[SK_{TD}, \text{Bob's public key is } PK_B]$
- **Replay Attack**: Bob changes keys, so he know has SK'_B and PK'_B. We imagine the MITM wants to intercept the TD saying PK'_B and replace it with PK_B. Alice must be able to determine it is the latest so we put a time stamp.
  - Alice says she wants PK_B and gives a nonce which she wants them to use. TD send back $Sig[SK_{TD}, \text{Bob's public key is } PK_B, nonce]$ This nonce shows the message was generated now, not earlier.
- Caveats: central point of attack (and failure), Scale-ability: it has everyone's PK, important to authenticate users to TD when updating PK, the service has to be online at all times during PK operation because you must fetch PK

2.1.2 Digital Certificates

- **Certificate**: a way to represent association between name and PK as certified by a third party (CA = certificate authority), like Verisign
- Using RSA: $Sign(SK_{CA}, \text{name=PK})$

- Alice obtains the certificate for PK_B from anyone and she knows she can trust it. This means it doesn't have to be online at all times (one advantage)
- Scale-ability also still a problem, although central point of failure no longer is
- **Certificate chains or hierarchical PKI**: The Gov. of CA Jerry can maintain around 100 certificates, including UCB president, who has his own 100 in turn, including Popa, etc. $Sign(SK_J, Presidential = PK_P)$
- If Alice wants to send a message to Nick - she asks people for certificate of Nick - then she gets back $Sign(SK_{UCBPresident}, Nick = PK_N)$. Then she must get the $Sign(SK_J, Presidential = PK_P)$ to verify the UCB president. Finally, everyone has hard-coded PK_J, so she can verify him as well. Now she knows that she received Nick's PK.

2.1.3 Revoking Certificate

:

- $Sign(SK_{CA}, PK_B = Bob$Experices on Nov 29, 2016 Checks that the client is still valid
- Use revocation lists: browsers put together of no longer valid keys that did not expire yet - black list

2.1.4 Attack Mechanisms

:

- Online guessing attacks - Attacker tries to login by guessing user?s password
- Social engineering and phishing - Attacker fools user into revealing password
- Eavesdropping - Network attacker intercepts plaintext password on the connection
- Client-side malware - Key-logger/malware captures password when inserted and sends to attacker
- Server compromise - Attacker compromises server, reads storage and learns passwords

2.1.5 Defense and Mitigations

:

- Network Eavesdropper: encryp traffic using SSL
- Client side malware: hard to defend - use two factor authentication
- For online guessing attacks: rate limiting: impose limit on number of passwords attempts, CAPTCHA: hard for robot to weird funny looking words in boxes, Password requirements
- For server compromise: Suppose attacker steals database at server including all passwords - don't store passwords in plaintext - Hash passwords instead using cryptographic hash function. When Alice logs in with pw check Hash(pw) = Hash(pw) stored on server to verify
- Password Hashing: with 2^20 passwords, you can guess 50 percent of passwords
- **Dictionary Attack**: attacker tries all passwords against hash(pw).
- Amortized password hashing: One brute force scan for many hashes. The attacker can check the hash for a word against every user's hash. We want to force the attacker to not be able to do this - want him to calculate the dictionary every time
- Randomize hashes with salt: server stores (salt, hash(password,salt)) - salt is random
  - Two equal passwords have different hashes now. This also means that they can't use the same hash for a given password because the salt is different for everyone user, meaning that the hash(password, salt) is different.
  - To make hashing slower (harder to brute force): we can use

    $$Hash(Hash(Hash....(Hash(pw)))))$$

    . But we don't want to do it too much or it will be too slow for the user

3 Applied Cryptography and Applied Craptography

3.1 Applied Cryptography

3.1.1 HMAC

- It is both a cryptographic Hash and a MAC

- XOR the key with the i_pad and zero extend the key so its one block wide if necessary

$$Hash(key_1 \oplus i_pad||\text{message})$$

$$Hash(key_2 \oplus o_pad||\text{first hash})$$

- The pads are slightly arbitrary (Hamming distance from each other far away) and the keys must be different
- The second hash prevents the attacker from just adding things onto the message to change the MAC

3.1.2 Facebook Messenger Abuse Complaints

- Facebook has an encrypted chat application on the phone now
- The crypto is good but not the point - used a library from Signal
- When Alice wants to send a message to Bob, queries FB for Bob's PK, encrypts m and send to FB who delivers it to Bob - out of order asynchronous delivery
- Both Alice and Bob are using encrypted and authenticated channels to FB
- Unique Messenger Problem: **Abuse**
  - If either Alice or Bob is mean or something - want to be able to handle complaints
  - Facebook can't necessarily trust Bob's complain about Alice - but the m is encrypted to FB can't just record them and read
  - Abusers might also want to release information publicly - so only FB should be able to verify an abuse complaint
  - Unless Bob forwards the unencrypted m to FB, FB must not know the contents - if Bob does FB must ensure this is the m that Alice sent to Bob - and nobody but FB should be able to verify
  - Facebook has a nearly hidden option which allows Alice to see Bob's key - if they ever get together they can manually verify that FB was honest - the mantra of key servers: **Trust but Verify**. iMessage for example can't be verified like this to ensure Apple's honesty
  - Alice ask's FB for Bob's PK (trusted key server). Alice creates a message that includes an abusive message and random value k_rand encrypted with Bob's public key. He also includes and HMAC(k_rand, m). Then FB only retains the HMAC of the m. Then Bob decrypts it. FB can not read the message or even verify Alice's HMAC, as the key for the HMAC is in the message itself. Only thing FB needs is its own HMAC key. Upon receipt, Bob checks that Alex's HMAC is

correct and rejects if not. Then Bob complains with the unencrypted m and sends back to FB, who can verify with the m to validate the complaint about the m sent from Alice to Bob and wasn't tampered with. verify $mac = HMAC(k_rand, M, k_rand)$ Bob couldn't prove to anyone other than FB because you need the FB key.

3.1.3 Generating Random Numbers

- RNG is heart of crypto - this is done in all kinds of systems
- Greater entropy using a pseudo random number generator
- TRUE random number requires a physical process using oscillators and physics things - other common sources are human activity like mouse movements or network activity. More exotic: a wall of lava lamps with a rotating prism and a camera
- Combining Entropy: use various sources measured in how many bits (1 coin flip = 1 bit)
- **Pseudo RNG** (Deterministic Random Bit Generators): if one knows the internal state, then you can predict - otherwise indistinguishable from a random string. (1) Seed: set internal state based on some real entropy sources (2) Reseed: Update the internal state based on prev state and additional entroy (3) Generate: a series of random bits based on internal state
- It should be **rollback-resistant** - can't run the generator backwards (even though you know all going forwards)
- $HMAC_DRBG$: believed to be the best PRNG - to break means you can break HMAC: two internal state registers V and K that are the same size as Hash functions output. V is used as data input to HMAC and K is the key. **Generate**: requires one HMAC call per blocksize (bits of state) - 3 times. Backtrack resistant (would require you to reverse the hash function). Prediction resistant - you would have to be able to determine HMAC from a random function. **Update**: 4 HMACs to take input and create change reseed. Used standalone for instantiate - use all our entropy sources and combine them. Designed so that even if the attacker knows input and not k, can't predict k.

3.2 Applied Craptography

3.2.1 Snake Oil

- **Snake oil** refers to fraudulent cures in the 19th century.
- Anti snake oil: NSA's CNSA cryptographic suite: unclass algorithms that are secure: AES256, SHA384, 3072 keys to RSA/Diffie Hellman, ECDHE/ECDSA: 384 keys (all in bits)

- Actual snake oil: Amazingly long key lengths - be suspicious - NSA only uses 256b keys for symmetric and 4096b for RSA/DH
- Actual snake oil: New algorithms and crazy protocols, One Time Pads (have to actually never reuse), new math, "chaos," "cracking contest" - decrpyt this message with no context and no structure.
-

3.2.2 Unusable Systems

Unusability: No public keys (everyone must have the same shared key) - what if someone doesn't have the days keys, PGP (Pretty Good Privacy) if you steal a key and have old conversation, you can decrypt old messages. PGP - also hard to hide metadeta and never transparent. PGP - how do you find someone's PGP key. openssl libcrypto and libssl - a bunch of different functions

3.2.3 Low entropy RNGs

- If you don't have enough - you can simply brute force and try every random number with 16 bits until you get it.

3.2.4 Sabotaged RNGs

- There were problems with an NSA random number generator. Were able to run $Dual_EC$ backwards and get the session key. Didn't even require a lot of computational power. The NSA had been using backdoors - someone rekeyed Junipers lock by changing P and Q - then someone ELSE added an ssh backdoor - then people noticed $theDual_EC$ parameters were changes also
- Sabotaging "Magic Numbers": these numbers are always suspicious. Parameters of an elliptic curve, points P and Q, and prime p for Diffie Hellman, the contents of S-boxes in block ciphers,
- Good systems should cleanly describe how their magic numbers things are generated - or use a pRNG with a specific seed (tell people it's seeded with this value) - this means people can trust it because if you couldn't someone would have to break the pRNG
- A good system should also be perceived as secure as well - otherwise people wouldn't use them.

4 Network Security

- Hacks to attempts prevent deficiency, and using protocols to mae underneath layers irrelevant

- The OSI 7 Layer Network Stack: Physical and Data Link (Ethernet and Wifi, DHCP and ARP), Network Layer (IP, DNS), Transport Layer (TCP and UDP, TLS, Firewalls), Session Layer, Presentation Layer, Application Layer (Network intrusion detection, leads into web security), Political Layer