ECE 435 – Network Engineering Lecture 6

Vince Weaver

https://web.eece.maine.edu/~vweaver

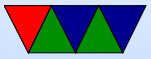
vincent.weaver@maine.edu

12:30pm, Barrows 125

1 February 2024

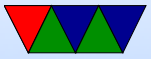
Announcements

- HW#3 will be posted. Encryption. No coding.
- HW#2 was extended to Friday
- If you send a e-mail wanting help with homework, be sure to make the e-mail subject reflect that (otherwise I might take longer to reply)



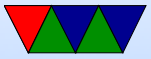
Asymmetric / Public Key Encryption

- Asymmetric/Public Key
- Key exchange is weakest link of symmetric encryption, as both sides need it and if it leaks, all is lost
- Have a public key that anyone can use to encrypt a message. Can only be (easily) decrypted by a secret, private key
- Hard to solve math problems. Integer factorization, discrete logarithm, elliptic curves



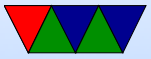
Why not use Asymmetric for everything?

- Often only used to encrypt small amounts of data, i.e. used to encrypt a symmetric key used for longer transactions
- High overhead and requires high-quality random numbers, hard to use it for large amounts of data



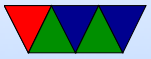
Uses of Public Key Crypto

- public key encryption
  - public key used to encrypt message only holder of private key can decrypt
- digital signature
  - message signed with private key and anyone with access to public key can verify the original sender

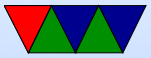


RSA

- Rivest/Shamir/Adleman at MIT (1977)
  Discovered before by UK govt (1973) but classified
- Choose two large primes p and q (1024+ bits)
- Compute: n=p*q, z=(p-1)*(q-1)
- Choose number relatively prime to z: d
  (no common factors)
- Find e such that e*d mod z=1
- Divide plaintext into blocks $0 \leq P < n$, blocks of k bits where k largest $2^k < n$

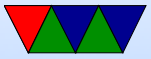


- To encrypt, compute $C = P^e \ mod \ n$
- To decrypt, compute $P = C^d \ mod \ n$
- public key is e,n. private key is d,n
- Hard to break as you need to factor n (hard)
- How do you find p and q? Generate random number, then apply various tests to determine if prime (there are algorithms for that)



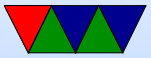
RSA Example

- Example from Tanenbaum Figure 8-17:
  Pick two large primes: p=3, q=11
  n=p*q=33, z=(p-1)*(q-1)=20
  d=7 (no common factors with 20)
  $7 * e \ mod \ 20 = 1$ so e=3
  private key=7,33 public key=3,33
  To encrypt "13", $13^3 = 2197, mod 33 = 19$
  To decrypt "19", $19^7 = 893871739 mod 33 = 13$

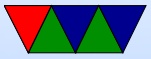


Why RSA Not Used Anymore

- Needs really good random primes, if you pick bad primes can be easier to crack (if p and q too close together)
- Slow, so on low-power devices tempting to pick low value exponents
- Adding more bits only slowly adds better encryption
- No random element, so can tell if the same message sent twice because will encrypt to the same (or can brute force easier)
  Fix to this is random padding at end

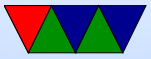


- Improper padding can lead to “padding oracle” attack (if you get an invalid padding error on invalid cyphertext, can slowly work your way to the key)

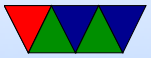


RSA Replacements

- RSA 2048 bit but even that might not be enough
- DSA (NIST 1991 / FIPS 1993)
  - built on modular exponentiation / discrete logarithms
  - Roughly same security with keysize as RSA
- ECDSA – elliptic curve cryptography (ECC) (1999)
  - Algebraic structure of elliptic curves on finite fields
  - Can provide same security with smaller keys than RSA/DSA
  - Endorsed by NSA

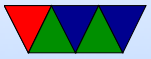


- 1024 bit RSA equivalent to 160 bit ECC



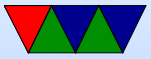
Cryptographic Hash Functions

- Maps a document of arbitrary size to a fixed size
- Easy to calculate, hard to reverse. Only real feasible way to reverse is brute-force search
- Break file up into chunks, do a series of operations to "compress" it, often shift, xor, or, add, and, not
- Small changes in document should lead to very different hashes



Hash Collisions

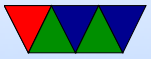
- Should not be able to find two different messages with same hash
- Two items with same hash are a collision
- Are collisions useful? If you can map documents of same filetype, or if somehow same document with lots of garbage on end



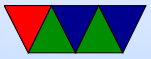
Cryptographic Hash Algorithms – md5

- md5 `md5sum` (Rivest) (1991, replacing md4)
- 128-bit md5 hashes, create checksum, almost uniquely ID file

  supposed to be unlikely to get collision
- Been broken, easy to defeat since 2007
  - Birthday attack, while creating two files with same sum hard, creating a huge number of files the likelyhood of getting two to be the same is more likely than you think

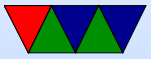


- Chosen-prefix attack – in this case take two differing start texts, by appending arbitrary data to each (in a comment section in some formats like PDF) can find match



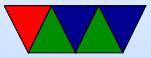
SHA-1

- Developed by NSA 1993
- 160-bits (40 hex digits)
- Deprecated by NIST since 2011
- SHAppening (2015)
- SHAttered (2017) first collision (pdf file)
- chosen-prefix attack 2019
- Used by git (oops)



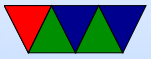
SHA-2, SHA-3

- SHA-2 (Secure-Hash Algorithm 2)
  - Designed by NSA, 2001
  - 224, 256, 384 or 512 bits
  - Merkle-Damgård construction
- SHA-3
  - Keccak, Sponge Construction
  - Different than others. Not meant to replace SHA-2 as SHA-2 not broken yet



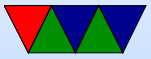
Cryptographic Hash Uses

- passwords (/etc/shadow)
- (mostly) uniquely identifying a file (git),
- verifying file contents (download, error checking),
- bitcoin?



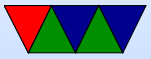
Proof of Concept —— GTFO

- One issue of hacker magazine had fun generating collisions
- Distributed as PDF that included its own md5sum (should that be hard?)
- Same PDF file was also a zip file and an NES ROM you could run in an emulator, also showing the sum



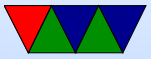
Other Encryption Concerns

- Redundancy, some way to validate plaintext is valid. Example: if encrypting a binary blob where each byte indicates something (12 34 means order 34 cows or something), random garbage might decode to valid message
- Freshness – replay attacks. What if you record old message (Bank deposits $100 to account) and replay. Will have valid encryption.

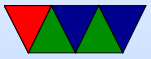


Encryption Problems

- Keys leaked (DVD/game console issues)
- poor random numbers used (Debian problem)
- differential cryptanalysis (start with similar plaintexts and see what patterns occur in output) [DES IBM/NSA story]
- Power/Timing analysis – note power usage or timing/cache/cycles when encryption going on, can leak info on key or algorithm
  Bane of perf

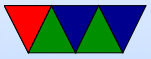


- Quantum computers



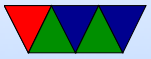
Trusting Trust

- When setting up an encrypted connection, how do you verify who is on the other side?
- How can you protect from man-in-the-middle attacks (MitM) where someone intercepts them downloading your public key, replaces with their own, then sits in the middle decrypting/re-encrypting in a transparent way?
- Some companies/countries will actually do this quite openly



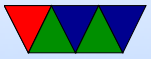
Key Signing Parties

- One way is to have get-togethers were friends sign each others keys
- If enough people do this, you can create a "chain of trust" where you can track someone's identity to someone you trust
- Linux kernel sorta tries this for git development
- Trouble for new people, or remote people, or people who don't travel much, or don't have many friends



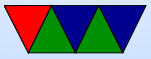
Self-signing Keys

- Can you just sign your own keys?
- You can, but how would someone know it's really you?
- (Though you could argue, does that always matter?)
- Most web browsers get very upset and will show lots of warnings if you use a self-signed key



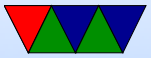
Certificate Authorities

- Certificate authority – an official organization that verifies identities
- Will sign a "certificate" saying who you say you are
- Operating Systems/Web-browsers will ship with a list of officially trusted Certificate Authorities
- Can hover over the lock symbol in URL bar to verify who signed for a website
- Hashed?
- Can be revoked



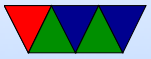
SSL/TLS

- Secure Socket Layer / Transport Layer Security
- Handshake protocol followed by key exchange
- Browser says hello, which hashes/algorithms it supports
- Server picks one and sends back
- Server then sends a certificate (signed by authority) saying who it is, and what its public key is
- Client verifies certificate (via the CA public key it has stored)
- client generates a random number, encrypts with servers



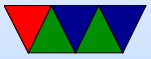
public key, sends to server, used as symmetric key

- What could go wrong, what if someone gets a hold of server private key? could decrypt logged data.
- Could try Diffie-Hellman key exchange – random number plus unique session key prevents problems if server private key leaked



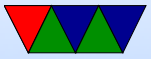
Diffie-Hellman (used by ssh)

- Both sides agree on large prime number
- Both sides agree on algorithm (AES?)
- Each side picks independently picks another secret prime number.
  This is not the authentication private key.
- The secret prime, AES, and shared prime are used to make a public key derived from the private key.
- The generated public key is shared
- The other side uses their own private key, the other side



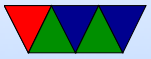
public key, and shared prime to figure out the shared secret key.

- This secret key is then used for symmetric encryption.
- Example on p812



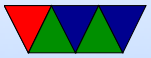
Encryption – Encrypting Hard Drives

- LUKS on Linux
- Bitlocker?
- This gets into the whole signed/trusted firmware on modern machines
- Ransomware

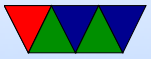


Encryption – Encrypting E-mail

- PGP – pretty good privacy
  OpenPGP RFC 4880
  Encrypt message with symmetric key, send along the key encrypted via asymmetric
  was illegal for a while (more than 40 bit encryption an exportable munition)
  people got RSA algorithm in perl tattoos
- GPG – free software replacement for PGP
- Can also PGP sign a message. Not encrypted, but signed



with your key to verify it was in fact sent by you. Takes hash of the input, then encrypts the hash with key. Also, downloads from servers (like debian)



Encryption – Ethics

- Should everyone be allowed to encrypt things?
- Should governments be allowed backdoors to decrypt things?