CSE 543 - Computer Security

Lecture 4 - Cryptography
September 6, 2007
URL: http://www.cse.psu.edu/~tjaeger/cse543-f07/
Hash Algorithms

• Hash algorithm
  – Compression of data into a hash value
  – E.g., \( h(d) = \text{parity}(d) \)
  – Such algorithms are generally useful in programs

• … as used in cryptosystems
  – One-way - (computationally) hard to invert \( h() \), i.e., compute \( h^{-1}(y) \), where \( y=h(d) \)
  – Collision resistant hard to find two data \( x_1 \) and \( x_2 \) such that \( h(x_1) == h(x_2) \)

• Q: What can you do with these constructs?
Hash Functions

• Design a “strong cryptographic hash function”
• No formal basis
  – Concern is backdoors
• MD2
  – Substitution based on pi
• MD4, MD5
  – Similar, but complex functions in multiple passes
• SHA-1
  – 160-bit hash
  – “Complicated function”
Message Authentication Code

• MAC
  – Authenticated integrity for data d
  – Uses some key k and hash algorithm h
  – To simplify,
    \[ \text{mac}(k,d) = h(k+d) \]

• Why does this provide integrity?
  – Cannot produce \text{mac}(k,d) unless you know k, d
  – If you could, then can break h
  – Exercise for class: prove the previous statement

• Used in protocols to authenticate content
HMAC

- MAC that meets the following properties
  - Collision-resistant
  - Attacker cannot compute proper digest without knowing K
    - Even if attacker can see an arbitrary number of digests H(k+x)

- Simple MAC has a flaw
  - Block hash algorithms mean that new content can be added
  - Turn H(K+m) to H(K+m+m’) where m’ is controlled by an attacker

- HMAC(K, d) = H(K + H(K + d))
  - Attacker cannot extend MAC as above
  - Prove it to yourself
Birthday Attack

- A birthday attack is a name used to refer to a class of brute-force attacks.
  - birthday paradox: the probability that two or more people in a group of 23 share the same birthday is > than 50%

- General formulation
  - function $f()$ whose output is uniformly distributed
  - On repeated random inputs $n = \{ n_1, n_2, \ldots, n_k \}$
    - $Pr(n_i = n_j) = 1.2k^{1/2}$, for some $1 \leq i, j \leq k, 1 \leq j < k, i \neq j$
    - E.g., $1.2(365^{1/2}) \approx 23$

- Q: Why is resilience to birthday attacks important?
Using hash values as authenticators

• Consider the following scenario
  • Alice is a teacher who has not decided if she will cancel the next lecture.
  • When she does decide, she communicates to Bob the student through Mallory, her evil TA.
  • She does not care if Bob shows up to a cancelled class
  • Alice does not trust Mallory to deliver the message.
• She and Bob use the following protocol:
  1. Alice invents a secret $t$
  2. Alice gives Bob $h(t)$, where $h()$ is a crypto hash function
  3. If she cancels class, she gives $t$ to Mallory to give to Bob
     – If does not cancel class, she does nothing
     – If Bob receives the token $t$, he knows that Alice sent it
Hash Authenticators

• Why is this protocol secure?
  – $t$ acts as an authenticated value (authenticator) because Mallory could not have produced $t$ without inverting $h()$
  – Note: Mallory can convince Bob that class is occurring when it is not by simply not delivering $h(t)$ (but we assume Bob is smart enough to come to that conclusion when the room is empty)

• What is important here is that hash preimages are good as (single bit) authenticators.

• Note that it is important that Bob got the original value $h(t)$ from Alice directly (was provably authentic)
Hash chain

- Now, consider the case where Alice wants to do the same protocol, only for all 26 classes (the semester)
- Alice and Bob use the following protocol:
  1. Alice invents a secret $t$
  2. Alice gives Bob $H_{26}(t)$, where $H_{26}()$ is 26 repeated applications of $H()$
  3. If she cancels class on day $d$, she gives $H^{(26-D)}(t)$ to Mallory, e.g.,
     - If cancels on day 1, she gives Mallory $H^{25}(t)$
     - If cancels on day 2, she gives Mallory $H^{24}(t)$
     .......
     - If cancels on day 25, she gives Mallory $H^{1}(t)$
     - If cancels on day 26, she gives Mallory $t$
  4. If does not cancel class, she does nothing
     - If Bob receives the token $t$, he knows that Alice sent it
Hash Chain (cont.)

• Why is this protocol secure?
  • On day $d$, $H^{(26-d)}(t)$ acts as an authenticated value (authenticator) because Mallory could not produce $t$ without inverting $H()$ because for any $H^k(t)$ she has $k>(26-d)$
  • That is, Mallory potentially has access to the hash values for all days prior to today, but that provides no information on today’s value, because they are all post-images of today’s value

  – Note: Mallory can again convince Bob that class is occurring by not delivering $H^{(26-d)}(t)$

• Important: chain of hash values are ordered authenticators

• Important that Bob got the original value $H^{26}(t)$ from Alice directly (was provably authentic)
Basic truths of cryptography …

• Cryptography is not frequently the source of security problems
  – Algorithms are well known and widely studied
    • Use of crypto commonly is … (e.g., WEP)
  – Vetted through crypto community
  – Avoid any “proprietary” encryption
  – Claims of “new technology” or “perfect security” are almost assuredly *snake oil*
Why Cryptosystems Fail

- In practice, what are the causes of cryptosystem failures
  - Not crypto algorithms typically
Case Study

• ATM Systems
  – Some public data
  – High value information
  – Of commercial enterprises, banks have most interest in security

• How do they work?
  – Card: with account number
  – User: provides PIN
  – ATM: Verifies that PIN corresponds to encryption of account number with PIN key (offset can be used)

• Foundation of security
  – PIN key (can obtain PIN if known and forge cards)
Simple Fraud

• Insiders
  – Make an extra card; special ops allow debit of any acct

• Outsiders
  – Shoulder surfing; fake ATMs; replay pay response

• PINs
  – Weak entropy of PIN keys; limit user PIN choices; same PIN for everyone

• User-chosen PINs
  – Bad; Store encrypted in a file (find match); Encrypted on card

• Italy
  – Fake ATMs; Offline ATMs (make several copies of card)
More Complex Issues

• PIN key derivation
  – Set terminal key from two shares
  – Download PIN key encrypted under terminal key

• Other banks’ PIN keys
  – Encrypt ‘working keys’ under a zone key
  – Re-encrypt under ATM bank’s working key

• Must keep all these keys secret
Products Have Problems

- Despite well understood crypto foundations, products don’t always work securely
  - Lose secrets due to encryption in software
  - Incompatibilities (borrow my terminal)
  - Poor product design
    - Back doors enabled, non-standard crypto, lack of entropy, etc.
  - Sloppy operations
    - Ignore attack attempts, share keys, procedures are not defined or followed
- Cryptanalysis sometimes
  - Home-grown algorithms!, improper parameters, cracking DES
Problems

• Systems may work in general, but
  – Are difficult to use in practice
  – Counter-intuitive
  – Rewards aren’t clear
  – Correct usage is not clear
  – Too many secrets ultimately

• Fundamentally, two problems
  – Too complex to use
  – No way to determine if use if correct
What Can We Do?

• Anderson suggests
  – Determine exactly what can go wrong
    • Find all possible failure modes
  – Put in safeguards
    • Describe how preventions protect system
  – Correct implementation of safeguards
    • Implementation of preventions meets requirements
  – Decisions left to people are small in number and clearly understood
    • People know what to do

• Problems of security in general
Important principles

• Don’t design your own crypto algorithm
  – Use standards whenever possible

• Make sure you understand parameter choices

• Make sure you understand algorithm interactions
  – E.g. the order of encryption and authentication
    • Turns out that authenticate then encrypt is risky

• Be open with your design
  – Solicit feedback
  – Use open algorithms and protocols
  – Open code? (jury is still out)
Building systems with cryptography

• Use quality libraries
  – SSLeay, lim (from Lenstra), Victor Shoup’s library, RSAREF, cryptolib
  – Find out what cryptographers think of a package before using it

• Code review like crazy

• Educate yourself on how to use library
  – Caveats by original designer and programmer
Common issues that lead to pitfalls

• Generating randomness
• Storage of secret keys
• Virtual memory (pages secrets onto disk)
• Protocol interactions
• Poor user interface
• Poor choice of key length, prime length, using parameters from one algorithm in another
A really good book on the topic