Lecture 4

Hashes
Hash

- takes arbitrary sized input, generates fixed size output
- cryptographic hash
  - one-way (computationally infeasible to find input for a particular hash value)
  - collision-resistant (can’t find two inputs that yield same hash)
  - output should look “random”
Uses

- Sign hash (digest) instead of message
- Store digests of files, to look for changes (e.g., viruses). (Tripwire does this) Why wouldn’t CRC work?
- With secret, can do anything a secret key algorithm can do (authenticate, encrypt, integrity-protect)
- irreversible pwd hash database
Authentication with Hash

both know secret K

Alice ——————————————————— Bob

I’m Alice

R

compare:

hash(R,K)

or:

I’m Alice, f(K,timestamp)?
Creating Encryption with Hash

• Create pad. First send IV in clear
  – $pad_1 = \text{hash}(K, IV)$
  – $pad_2 = \text{hash}(K, pad_1)$
  – $pad_i = \text{hash}(I, pad_{i-1})$

• Note, with IV, Alice can precompute pad, but Bob can’t
Create a hash with a secret key

• Want it to be irreversible
  – infeasible to find a message with a particular hash
  – e.g., with passwords, be able to verify hash of a pwd without being able to discover the pwd from the hash

• How would you do this?
What if msg large? Will this work?

constant

\[ \text{m}_1 \rightarrow \text{e} \]

\[ \text{m}_2 \rightarrow \text{e} \]

\[ \text{m}_i \rightarrow \text{e} \rightarrow \text{hash} \]
Remember meet-in-middle attack

- If hash is $n$ bits, this allows work of $2^{n/2}$ to find a message with a particular hash
- fix by XOR’ing input of previous stage with output (figure 5-2 from book)
- Has same effect as using $K_1=K_3$ in 3DES. Constrains it sufficient to avoid MITM
“Random”

- Each hash value seen in practice should have about 1/2 the bits on
- Changing one bit out input should change about 1/2 the bits (unpredictable which)
- Two outputs should be uncorrelated, regardless of how closely related the inputs
- any subset of the bits should be a good hash
How big does it need to be?

- To find two messages with same $n$-bit hash, probably need to try square root of $2^n$ msgs, i.e., $2^{(n/2)}$
- If given one message, would take $2^n$ msgs about to find another with same hash
- Some uses don’t need such collision resistance
Salt

- Protects a database of hashed passwords
- Salt is non-secret, different for each user
- Store hash(pwd, salt)
- Users with same pwd have different hashes
- Prevents intruder from computing hash of a dictionary, and comparing against all users
Threats

• information gained through:
  – eavesdropping
  – server database theft
  – impersonation of Bob
  – impersonation of Alice

• dictionary attacks, directly obtaining secret
Lamport’s Hash (S/Key)

Bob’s database holds:

\[ n, \text{salt}, \text{hash}^{n+1}(\text{pwd} \mid \text{salt}) \]

Alice

Bob

I’m Alice

\[ n, \text{salt} \]

\[ \text{hash}^n (\text{pwd} \mid \text{salt}) \]
Lamport’s Hash (S/Key)

- Offers protection from eavesdropping and server database reading without public key cryptography
- No mutual authentication
- Only finitely many logins
- Small n attack: someone impersonates Bob
MACs with hashes

• Combine message with key and digest that
• Collision resistance isn’t important here. (why?)
Possible problem

- If do hash(key | message) and use entire result as MAC, some hash algorithms (SHA-1, MD5, MD4) have flaw
- HMAC proven not to have this problem
- HMAC is more work
  - 3 extra blocks if no precomputation
  - 1 extra block if precompute based on reuse of same key
The problem

- Some (most) hash algorithms can continue from where they left off.
- So even if you didn’t know the key, if you knew hash(key | message) you could continue
- Other solutions would have worked. Suggestions?
If others, why HMAC

• HMAC comes with a proof
  – assuming underlying function is
    • collision resistant
    • if attacker doesn’t know $K$, cannot compute proper
      digest($K,x$) even if sees arbitrary ($y,digest(K,y)$)
• Others likely just as secure, but no “proof”
Unix Password Hash

- They were so proud of its being irreversible they made the database of hashed passwords world-readable
- Password-guessing (dictionary) attacks
- (does S/Key also have this?)
- `crypt(8-char pwd, salt)` uses modified DES
  - encrypt 0 with pwd
What about too long pwds?

• could ignore all but 1st 8 chars
• typical: store crypt(1st 8 bytes), crypt(2nd 8 bytes)
  – what’s wrong with this?
Cookie Story

- Fu, Sit, Smith, Feamster collected web cookies (Usenix Security, 2001)
- Noticed cookie the same for all usernames with same prefix (with prefix > 7 chars), and indeed was crypt(username)
- Hypothesized crypt(username|server secret)
- Found the server secret!
Another Use of Hashes

• Micali Revocation Scheme
• In certificate, include two quantities:
  – hash^n(constant1)
  – hash(constant2)
• Every “day” (...revocation interval) if cert still valid, reveal one less hash of constant1 to user. Enables user to prove not revoked
• If revoked, reveal constant2 and post it
Proposed MAC for multicast

- MAC first packet with hash\(^n\)(secret), where secret known only to sender
- MAC 2nd packet with hash\(^{n-1}\)(secret), and include in packet hash\(^n\)(secret)
-...
- Why I didn’t like it…
- How it was modified
Multicast Key Distribution

- Assumption: have a traffic stream that lots of members are authorized to read (like pay TV)
- Want to change key if membership changes
- Assume each member has its own key (perhaps a public key pair)
- Inefficient to tell each member the new key individually each time it changed
Straightforward way

• Individually tell n members new key, O(n)
• But can send a message that all of them will receive (this is multicast, after all)
• What if send DEK (data encrypting key) encrypted with K1, and with K2, where half the members have K1 and half have K2?
Another way

• Iolus
  – have regions, each with its own key. Have node on boundary know both keys, and decrypt and reencrypt. How would it do this efficiently? What is the advantage of this if the group’s membership is very dynamic?
Another way

• Briscoe (NGC 1999)
• Change key frequently and periodically
Hash trees

- Suppose you want to send lots of pkts, and digitally sign them
- And some may get lost
- And you only want to bother with doing one signature