Crypto with Passwords

Lecture 22

Passwords

- Password or passphrase: Low-entropy shared secret
  - Typical goal: client authenticating to server, without being tied to a device holding a cryptographic key. On authentication, set up a session key.
  - Also, often Mutual Authentication (if server/client can't/doesn't want to use certificates to verify server's authenticity)
- Cannot get "negligible" security error: password can be guessed with some significant probability
- Goal: allow only an online guessing (dictionary) attack. Prevent offline dictionary attacks.
  - Or if server compromised, still somewhat protect the passwords, by allowing only a slow offline dictionary attack

Key from Password

- Common scenario: client only has a password rather than a key. Server has some information derived from client's password
- They will on-the-fly generate a session key from the password, and interact using it
  - Note: Client may not a priori know if the server is genuine
- Requires the key to be as good as random, up to the probability that the adversary can guess the password and interact with the server itself
  - Rate/number of attempts would be limited, so online dictionary attack would be OK
- Simple solution in the random oracle model: Key = H(passwd)
  - Note: Standards here often call H a "PRF" rather than hash, as not only collision resistance, but also pseudo randomness is important

Key from Password

- Simple solution in the random oracle model: Key = H(passwd)
- But if the password server is compromised an attacker can launch an offline dictionary attack
  - Typically quite feasible to discover many passwords
  - Attacker may possess a "Rainbow Table" — precomputed hashes of a dictionary — and can quickly recover almost all the stored passwords
- Typical solutions
  - **Salting** prevents Rainbow Table attacks: Key = H(passwd,salt) where salt is a long random string (sent to the client)
  - To make offline dictionary attack harder, use (moderately) hard hash functions

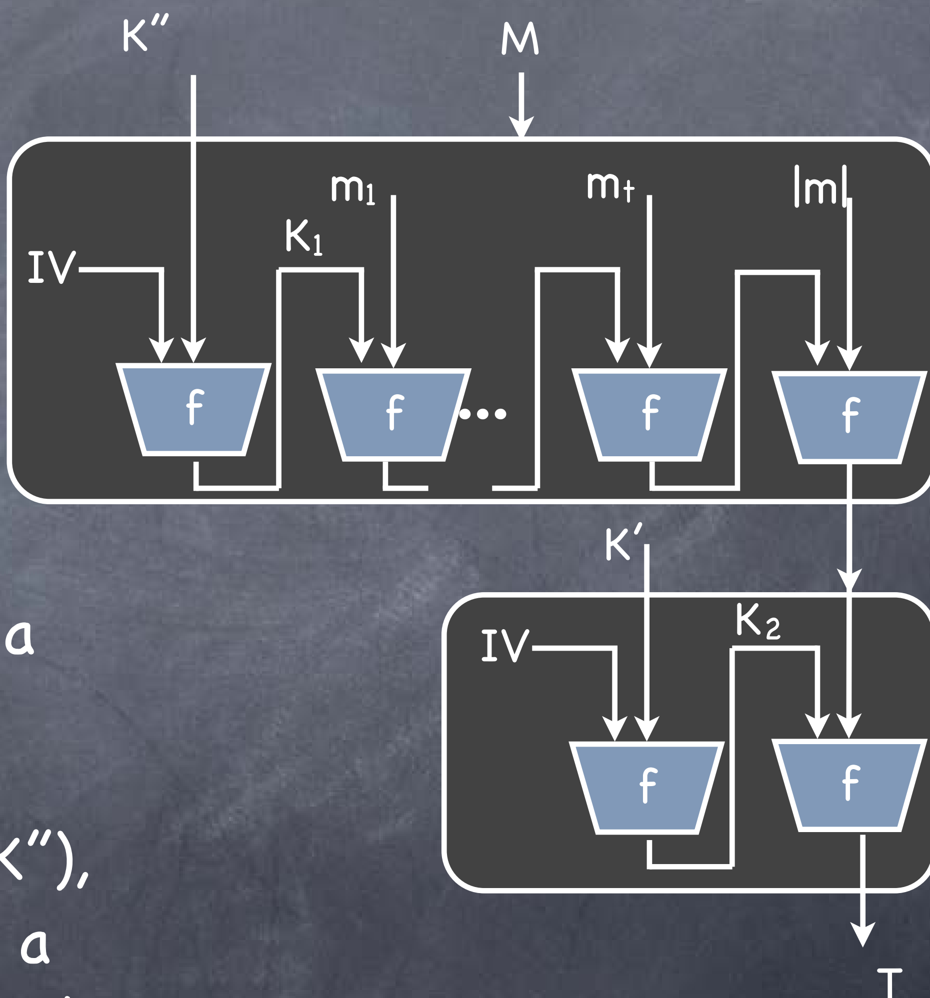
Key from Password

- Idea: computing $H(\cdot)$ should be moderately hard, so that the attacker is slowed down
- Iterated hash functions
  - e.g., PBKDF2 in RSA PKCS #5 (version 2):
    H(IV,msg) treated like a PRF, with IV being a key.
    Iterate as $U_1 = H(\text{Passwd},\text{Salt})$, $U_{i+1} = H(\text{Passwd},U_i)$.
    Output length extended using "counter mode".
- WPA2: between an Authenticator (server) and a Supplicant (client), where they share a "Pre-Shared Key":
  PSK = PBKDF2(hash = HMAC-SHA1, #iterations = 4096,
  msg = Passwd, salt = SSID, output length = 256)
  "Transient Key" derived from PSK, nonces, and mac addresses.
  Only nonces are exchanged between server & client.

RECALL

HMAC

- HMAC: Hash-based MAC
- Essentially built from a compression function f
  - If keys K_1, K_2 independent (called NMAC), then secure MAC if: f is a fixed input-length MAC & the Merkle-Damgård iterated-hash is a weak-CRHF
  - In HMAC (K_1,K_2) derived from (K',K''), in turn heuristically derived from a single key K. If f is a (weak kind of) PRF K_1, K_2 can be considered independent



Key from Password

- While iterated hashing slows down attack in software, much faster custom hardware (a.k.a ASIC) is not too expensive
  - Solution (on going research): Memory Hard Functions
    - Fast memory is still very expensive
    - So try to make the function require large amounts of memory.

Key from Password

- No forward secrecy in WPA2!
- If password is revealed past sessions can be decrypted
    - Transient key is derived from password and publicly known values (nonces exchanged)
    - Solution: Use keys from password only for authentication and use key exchange to derive encryption keys
    - Password-Authenticated Key Exchange (PAKE)

PAKE

- Password-Authenticated Key Exchange
  - Agree on a secret symmetric key, over a network
  - Client has a password, and server has related information
- Some considerations
  - A session is compromised if the session key is not pseudorandom to the adversary
  - Adversary can interact with the server, or with the client, or with both, concurrently in multiple sessions that use the same password (MITM attacks)
  - Adversary may learn a session key in one session, but that shouldn't compromise the keys in other sessions
  - Adversary may corrupt the client or sever (learning password information), but this shouldn't compromise past sessions

PAKE Protocols

- Several constructions, starting in early 90's, providing varying levels of security
- Typical construction uses H(passwd) to mask a DDH key-exchange
  - Due to DDH security, eavesdropping adversary doesn't learn the key
  - Without password, an adversary playing as client/server doesn't learn the key accepted by its honest partner
  - Example: Server given (v,s) to store, where $v = g^{\pi}$, $\pi = H(s,pwd)$. client→server: g^x ; server→client: u, $v+g^y$ (i.e., v as a mask); $K = (g^y)^{x+u\pi} = g^{xy} \cdot g^{u\pi y} = g^{xy} \cdot v^{uy}$. Key = H(K).
    - Client needs to know π. Note: without u, attacker knowing v alone can succeed by sending g^x/v in the first step

PAKE Protocols

- Protocols currently used in practice are proven secure in the random oracle model (under multiple security definitions)
  - Standard model protocols are known
- Need more comprehensive definitions to address concerns of composition: e.g., when multiple (related) passwords are used with multiple servers
- Universally Composable security (REAL/IDEAL security definition)
  - In the IDEAL world, a trusted party comparing passwords provided by parties, and if equal, allocating them random keys. Note: Even in IDEAL, security depends on passwords.
- But not realisable without a setup (e.g., random oracle, or common random string)