CS 5430

Passwords, part 2

Prof. Clarkson
Spring 2017

Review: Authentication of humans

Categories:

- Something you know

  password, passphrase, PIN, answers to security questions

- Something you have

  physical key, ticket, {ATM,prox,credit} card, token

- Something you are

  fingerprint, retinal scan, hand silhouette, a pulse

Review: Password lifecycle

1. **Create:** user chooses password
2. **Store:** system stores password with user identifier
3. **Use:** user supplies password to authenticate
4. **Change/recover/reset:** user wants or needs to change password

2. PASSWORD STORAGE

Review: Salted hashed passwords

- Defend against offline guessing attacks
- Each user has:
  - username uid
  - unique salt s
  - password p
- System stores: uid, s, H(s, p)

To authenticate Hu to L:

```
1. Hu->L: uid, p
2. L: let h = stored hashed password for uid;
      let s = stored salt for uid;
      if h = H(s, p)
      then uid is authenticated
```

Review: Salt

To combine with iterated hashing, include salt in first hash:

```
z1 = H(p, s);
z2 = H(p, z1);
...
...
z10000 = H(p, z9999);
output z1 XOR z2 XOR ... XOR z10000
```

this idea used in widely-deployed algorithm for deriving encryption keys from passwords...

PBKDF2

- Password-based key derivation function 2 [RFC 8018]
- **Output:** derived key k
- **Input:**
  - Password p
  - Salt s
  - Iteration count c
  - Key length len
  - Pseudorandom function PRF: "looks random" to an adversary that doesn't know an input called the *seed*

PRF

Common instantiation is HMAC

- PRF(m; s) is thus HMAC(m; s)
- That is, seed of PRF becomes key of MAC

PBKDF2

Algorithm:

- k = T(1) || T(2) || … || T(n)
  - enough T's to achieve desired len
  - || denotes bit concatenation
- T(i) = F(p, s, c, i)
  - F is in essence a salted iterated hash…
- F(p, s, c, i) = U(1) XOR … XOR U(c)
  - U(1) = PRF(s, i; p)
  - U(j) = PRF(U(j-1); p)

PBKDF2

- Could use to store passwords
- F(p,s,c,1) is essentially what we agreed to store for salted and iterated-hash protected passwords

WiFi

- WiFi WPA2 uses PBKDF2
  - Derive long-term key from passphrase
  - (later derive session keys from long-term key)
- Long-term key derivation:
  - p = passphrase
  - s = ssid
  - c = 4096
  - len = 256

WiFi

- Long-term key derivation:
  - PRF = HMAC with SHA-1
    - WPA2 is from (pre-)2004
    - SHA-1 attacks didn't appear until 2005
    - Protocols live forever!
    - *"During 2004 and 2005, there were a number of attacks on SHA-1 that reduced its perceived effective strength against collision attacks…However, since these attacks centered on finding collisions between values, they are not a direct security consideration here because the collision-resistant property is not required by the HMAC authentication scheme."* [RFC 8018]

1. PASSWORD CREATION

Who creates?

- **User:** typically guessable passwords
- **System:**
  - can produce hard-to-guess passwords (e.g., random ASCII character strings)
  - but users can't remember them
- **Administrator:** reduces to one of the above

Weak passwords

Top 10 passwords in 2015: [SplashData]

1. 123456
2. password
3. 12345678
4. qwerty
5. 12345
6. 123456789
7. football
8. 1234
9. 1234567
10. baseball

21: princess, 23: solo, 25: starwars

Top 20 passwords suffice to compromise 10% of accounts [Skyhigh Networks]

Strong passwords

- How to characterize strength?
- Difficulty to brute force—"strength" or "security level"
  - Recall: if 2^X guesses required, strength is X
- Suppose passwords are L characters long from an alphabet of N characters
  - Then N^L possible passwords
  - Solve for X in 2^X = N^L
  - Get $X = L \log_2 N$
  - This X is aka entropy of password
    - Assuming every password is equally likely, X is the *Shannon entropy of the probability distribution* (cf. Information Theory)

Entropy of passwords

NIST (2006) recommends:

- minimum of 14 bits
- but 30 bits more reasonable
- How does that work out in practice...?

Entropy of passwords

- Option A (toward 30 bits):
  - 8 character passwords chosen uniformly at random from 26 character alphabet
  - entropy of $8 \log_2 26 \approx 37$ bits
  - but that means abcdefgh equally likely as ifhslgqz
- Option B (toward 14 bits)
  - 1 word chosen at random from entire vocabulary
  - average high-school graduate: 50k word vocabulary
  - entropy of $\log_2 50k \approx 16$ bits

Entropy estimation

- **Problem:** guide users into choosing strong passwords
- Entropy estimates [NIST 2006 based on experiments by Shannon]:
  - (assuming English and use of 94 characters from keyboard)
  - 1st character: 4 bits
  - next 7 characters: 2 bits per character
  - characters 9..20: 1.5 bits per character
  - characters 21+: 1 bit per character
  - user forced to use lower & upper case and non-alphabetics: flat bonus of 6 bits
  - prohibition of passwords found in a 50k word dictionary: 0 to 6 bits, depending on password length

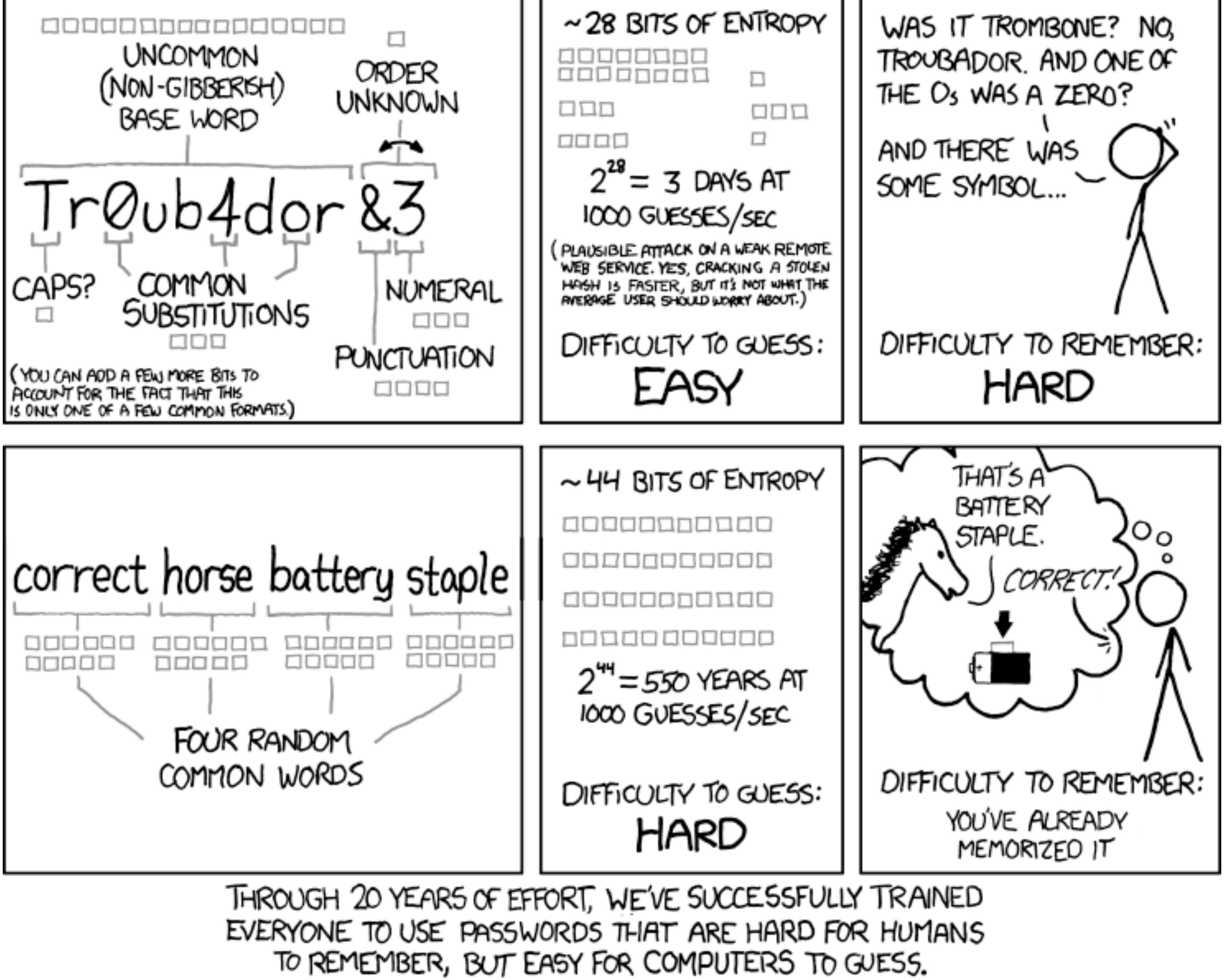
Entropy estimation

But:

- [Weir et al. 2010] based on cracking real-world passwords conclude *"[NIST's] notion of password entropy...does not provide a valid metric for measuring the security provided by password creation policies."*
- Underlying problem: Shannon entropy not a good predictor of how quickly attackers can crack passwords

Password recipes

- Recipes: rules for composing passwords
  - e.g., must have at least one number and one punctuation symbol and one upper case letter
- Naively seems wise
- But research suggests...
  - Users who are annoyed by recipes chose weaker passwords
  - Users pick easy-to-guess passwords that minimally comply with the recipe
- Beyond recipes?
  - After user picks password, system inserts mandatory randomness into it (which users must remember): users start choosing weaker base passwords
  - Password wallets: can have random passwords, but users have to trust storage of all their passwords to a program or service
  - Longer memorable passphrases...

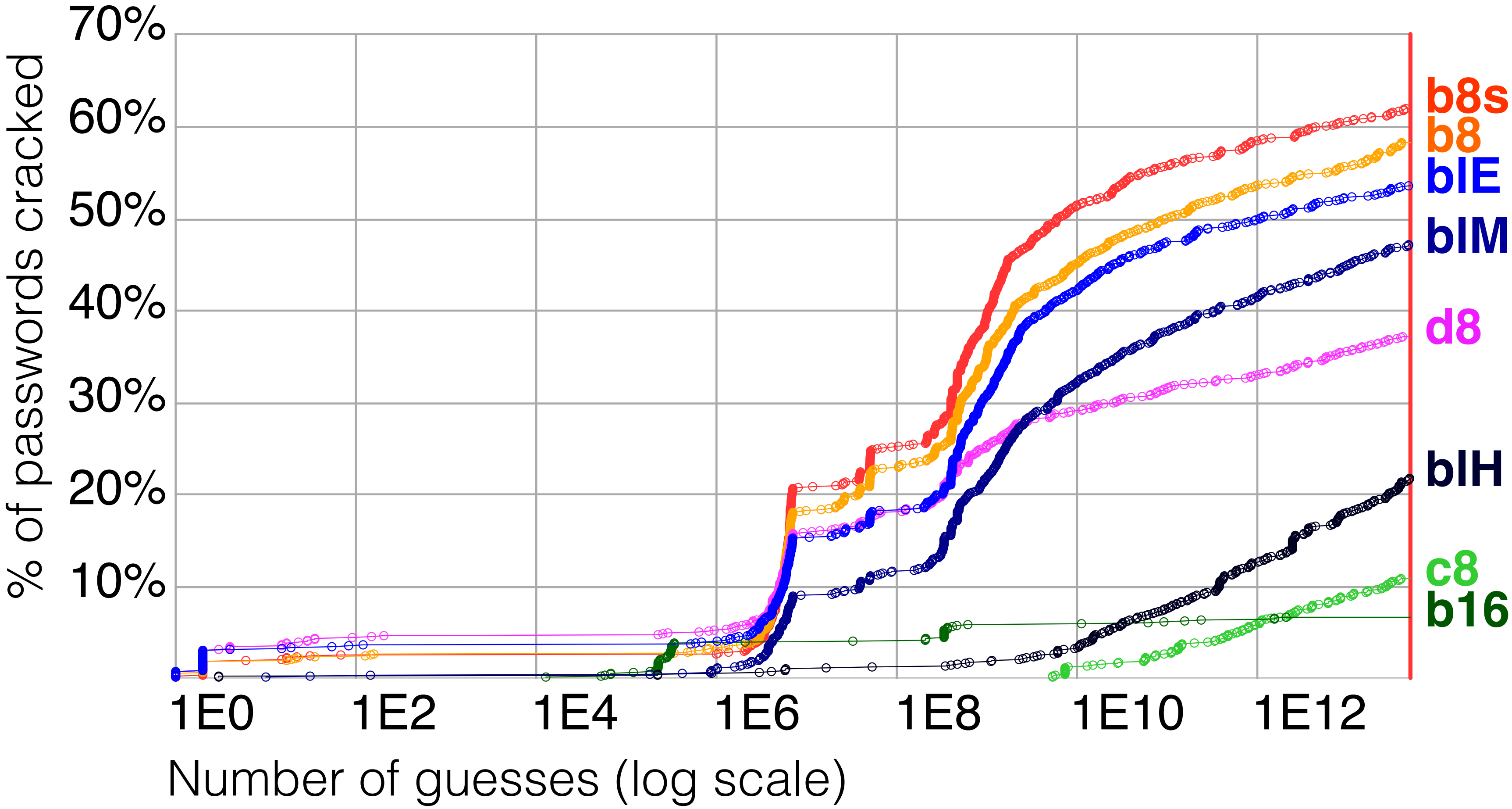

UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor&3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^{28} = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS:
EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER:
HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^{44} = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS:
HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER:
YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Recipe comparison

[Kelley et al. 2012]

- Evaluate recipes based on
  - percentage of passwords cracked
  - number of guesses required to crack
  - for two state-of-the-art cracking algorithms, one of which is from [Weir et al. 2010] (same paper that invalidates Shannon entropy)
- Selected recipes:
  1. ≥ 8 characters
  2. ≥ 8 characters, no blacklisted words ...with various blacklists
  3. ≥ 8 characters, no blacklisted words from freely available 4M word common password + dictionary word list, one uppercase, lowercase, symbol, and digit ("comprehensive", c8)
  4. ≥ 16 characters ("passphrase", b16)
- Results...

Recipe comparison



Recipe comparison

- Comprehensive recipe (c8) makes it hard to crack passwords
  - Doesn’t that contradict [Weir 2010]?
  - No: even if NIST's Shannon entropy estimates are quantitatively invalid **in general**, c8 in particular is hard to crack
- But passphrases (b16) aren't that much easier to crack
  - Threat to validity: maybe state-of-art crackers would improve to handle passphrases if people were required to use them
- And passphrases are more usable [Komanduri et al. 2011]:
  - Easier to create
  - Easier to remember

Password lifecycle

1. **Create:** user chooses password
2. **Store:** system stores password with user identifier
3. **Use:** user supplies password to authenticate
4. **Change/recover/reset:** user wants or needs to change password

Beyond passwords?

- Passwords are tolerated or hated by users
- Passwords are plagued by security problems
- **Can we do better?**
- Criteria: [Bonneau et al. 2012]
  - Security
  - Usability
  - Deployability

…criteria are worth studying for security in general

Security

- Physical observation
- Targeted impersonation
- Online guessing
- Offline guessing
- Internal observation
- Leaks
- Phishing
- Theft
- Trusted third party
- Privacy

Usability

- Memoryless
- Scalable for users
- Nothing to carry
- Physically effortless
- Easy to learn
- Efficient
- Infrequent errors
- Easy recovery from loss

Deployability

- Accessible
- Cost
- Server compatible
- Browser compatible
- Mature
- Non-proprietary

Schemes to replace passwords

- Password managers
- Proxies
- Federated identity management
- Graphical
- Cognitive
- Paper tokens
- Visual cryptography
- Hardware tokens
- Phone-based
- Biometric

Schemes to replace passwords

[Bonneau et al. 2012]:

- Most schemes do better than passwords on security
- Some schemes do better and some worse on usability
- Every scheme does worse than passwords on deployability
- **Passwords are here to stay, for now**
- Schemes offering some variation of single sign on seem to offer best improvements in security and usability...

Single sign on (SSO)

- The world without SSO: User enrolls with many service providers (SP), shares authentication secrets, e.g. password, with each
  - common scenario: user registers same or predictably modified password with each SP
  - products even exist to automatically synchronize passwords across SPs
    - usability trumps security
- With SSO: user authenticates **only once** with SSO service, thereafter SSO manages authentication to SPs
  - user has potentially multiple identities with SSO
  - user has potentially multiple identities with SPs
  - SSO trivially can impersonate user

Single sign on

Varieties of SSO: [Pashalidis and Mitchell 2003]

- Pseudo SSO: user authenticates to SSO, it uses SP's own authentication mechanism to authenticate on behalf of user
- True SSO: user authenticates to SSO, it **asserts** user's identity to SP

Pseudo SSO

- User selects identity to authenticate to SSO
- SSO stores user's secrets to authenticate on behalf of user to a particular identity at SP
- Local vs. remote/proxy...

Local pseudo SSO

- SSO service is local to user's machine
  - Typically an encrypted password DB
  - Degree of automation might vary
- Example: password managers
- Since SSO service must present user's secrets to SP, user must trust SSO & local machine with those cleartext secrets

Proxy pseudo SSO

- SSO service is on remote server
  - Typically fully automated, even invisible to user
  - Authentication to SP is redirected (e.g. HTTP 302) to or intercepted by proxy
- Local machine doesn't have access to secrets, but remote service is trusted with cleartext secrets
- Closest example: web browsers with auto-fill and cross-machine synch capabilities

True SSO

- User selects identity to authenticate to SSO
- SSO asserts user's identity to SPs
  - SP is being notified of authentication rather than deciding itself
  - notion of identity might vary between SPs
- Local true SSO: SSO is under physical control of user (less common)
  - could build off of trusted cryptographic co-processor to take control away from user
- Proxy true SSO: external proxy brokers between users and SP
  - Examples: Kerberos, Microsoft Passport

Difficulty of proxy true SSO

- SSO and SPs have a trust relationship, supported by
  - agreement about rights and responsibilities
  - secure communication channel
  - countermeasures to ensure SSO is not compromised
- SSO must define:
  - uniform meaning for attributes used in identities
  - accuracy standard for attributes
  - how to exchange and store secrets
  - obligations of SPs
  - legal instrument for accountability

Upcoming events

- [today] A4 out

Humans are…large, expensive to maintain, difficult to manage, and they pollute the environment. It is astonishing that these devices continue to be manufactured and deployed. But they are sufficiently pervasive that we must design our protocols around their limitations.

– Kaufman, Perlman, and Speciner