HumanAUT

Secure Human Identification Protocols

Adam Bender

Manuel Blum

Nick Hopper

The ALADDIN Center

Carnegie Mellon University

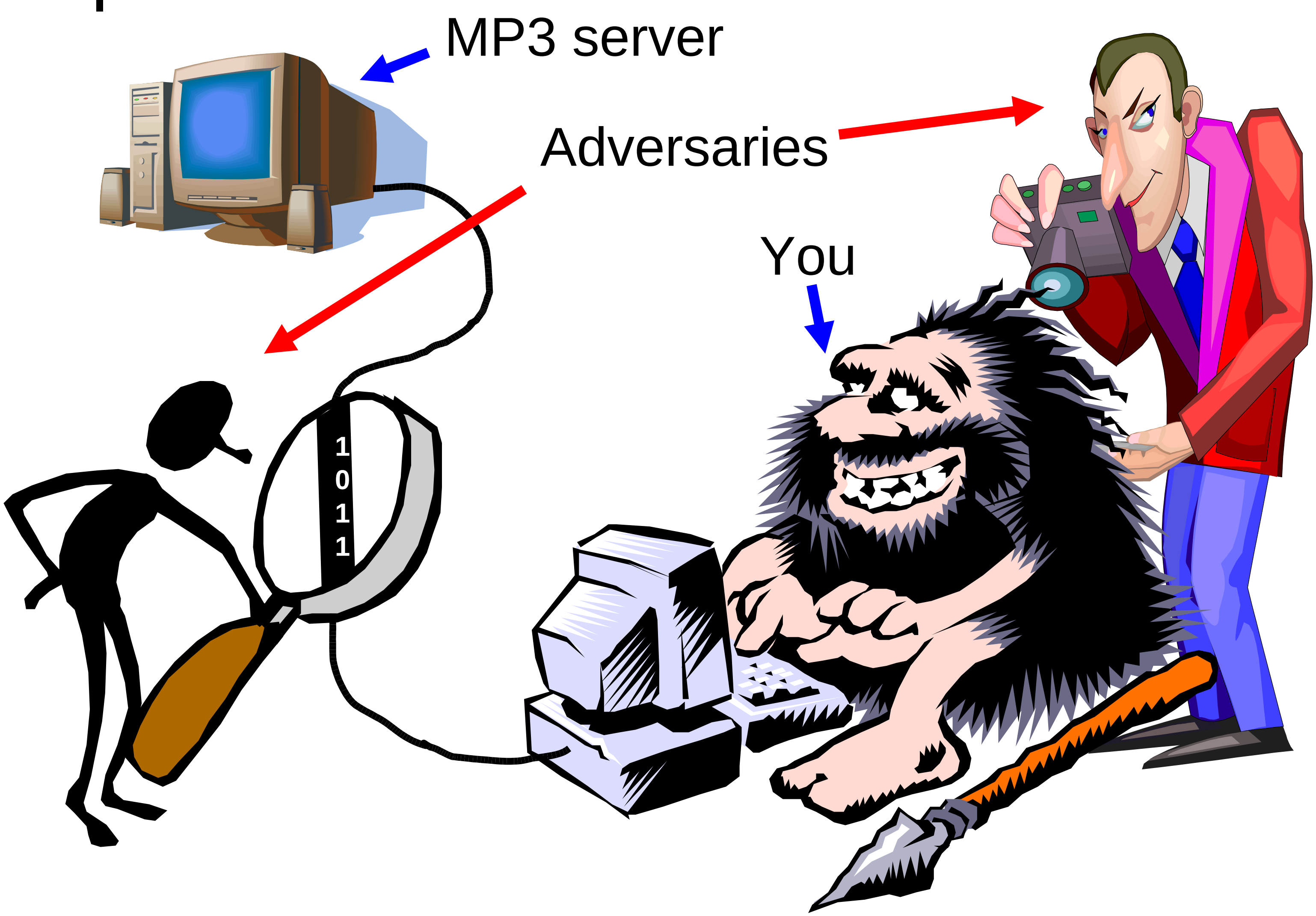
What is HumanAUT?

- HumanAUT stands for Human AUThentication
  - Authentication: proving your identity (logging in) to a third party
- Protocols for a human to authenticate to a computer/ATM/etc.
- With a few twists…

A hypothetical environment

- Assume you are a:
  - Naked person
  - In a glass house
  - With an insecure terminal
- Or you have lost your luggage, or had your wallet stolen, etc.
- How do you authenticate yourself?
- HumanAUT attempts to solve this

A hypothetical environment



Motivation: conventional authentication methods

- Many authentication methods are currently in use
- Passwords, PIN numbers, smart cards, tokens, biometrics

- Each one of these has problems!

Conventional authentication – problems (1)

- Passwords can be snooped / key-logged / sniffed / surveilled / cracked / intercepted / guessed / dumpster dived / etc.
- PIN numbers are short and often easy to guess (birthdays), susceptible to “shoulder surfing”

Conventional authentication – problems (2)

- Hardware is expensive and relies on physical mechanisms, which can be stolen
- Biometrics are also expensive, and not as secure as we thought
  - Gelatin fingers (Matsumoto ‘02)
  - Able to reconstruct sample images from face recognition template (Adler ’03)

Conventional authentication – problems (3)

- Once someone steals your smart card or lifts fingerprints, it is very expensive or impossible to change your secret – single use access methods are better
- One-time passwords are very inefficient though – ideally, want reusable challenge-response system

Challenge-response

- Computer asks a series of questions
- If answered correctly, user is authenticated, otherwise they are locked out
- Think of “Password: ” prompt as challenge, password as response
- But with different questions and answers each time

HumanAUT

- HumanAUT is a challenge-response scheme with a shared secret between the human, who answers challenges using the secret, and the computer, which generates a unique random challenge on demand, where the correct response depends on the shared secret

Goals

- Secure
  - No one else can authenticate themselves, even after observing successful authentications
- Human executable
  - People have to do it in their heads – no hardware or other aids

Definitions (1)

- Authentication protocol
  - Two sets of instructions, *H* (human) and *C* (computer), with a shared secret *s*
  - *C* generates random challenges for *H* to respond to, and decides to accept or reject based on *H*'s response
  - If *H* and *C* are using the same *s*, *C* accepts with high probability, otherwise rejects with high probability.
  - Assume presence of adversary *A*

Definitions (2)

- Passive adversaries
  - Observes authentications, then tries to authenticate
- Active adversaries
  - May construct messages and present them to *H*
- Note that if *A* can intercept a challenge, have *H* respond, and then present that to *C* (man-in-the-middle attack), no protocol can be secure!

Definitions (3)

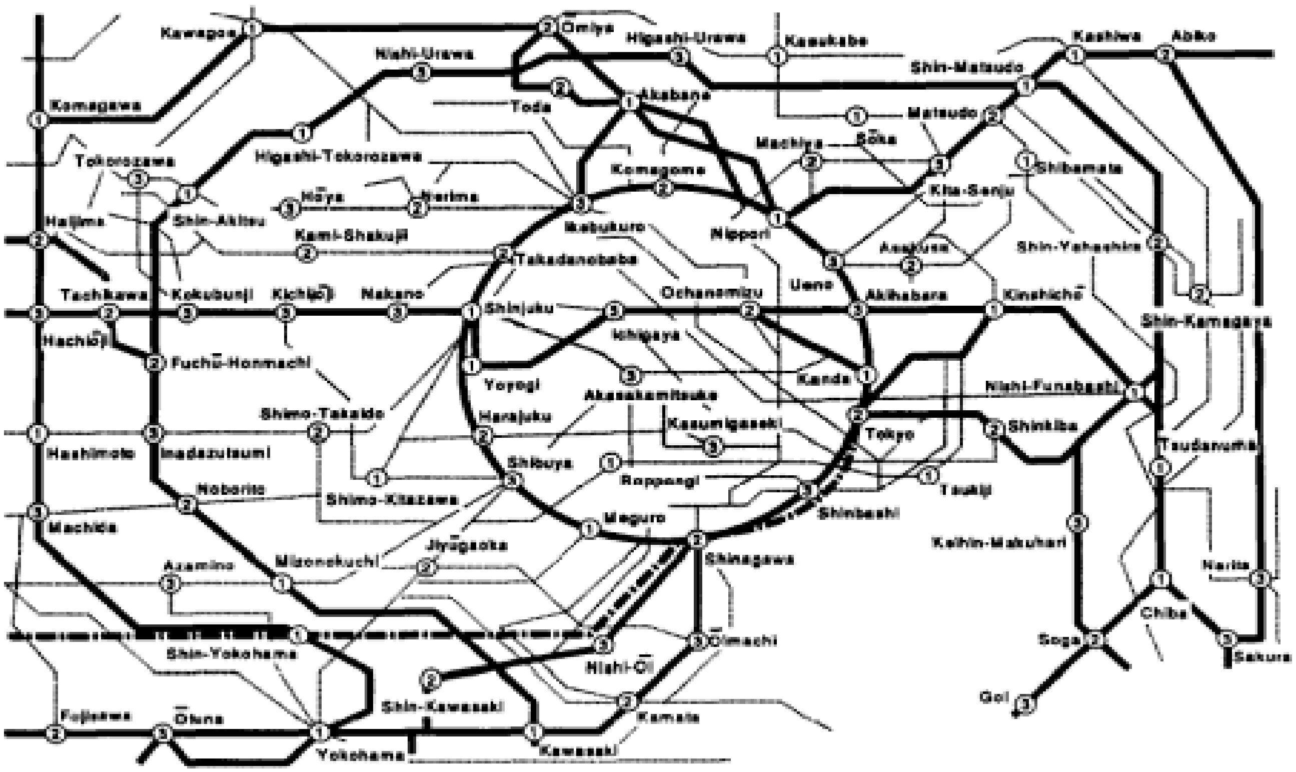
- A protocol is:
  - (p, k) secure if, for polynomial-time adversary A observing k authentications, P(C accepts A) $< p$
  - (p, q, k) detecting if it is (p, k) secure and if H can detect invalid challenges with probability $> 1-q$

Previous work - Matsumoto

- Matsumoto has come up with many schemes
- First was based on hiding a secret string among random characters, in positions dependent on the challenge
- Later broken, proposed new schemes

Previous work - Matsumoto

- Including map-based scheme



Previous work - Déjà Vu

- Perrig created Déjà Vu, a graphical recognition based system
- Based on recognition of images, not recall of secret key
- Requires large amount of graphical overhead – creation, storage, presentation
- Only good as memory aid – no real security guarantees as implemented

Previous work - Blum

- Mapping from letters $\rightarrow$ digits
- Challenge is an English string, response is a string of digits, each letter in the challenge corresponds to one digit in the response
- Letters transformed into digits and arithmetic manipulation performed with them
- s = mapping, arithmetic function

Previous work – Hopper (1)

- Created many of the useful definitions
- Scheme of filling a large grid with blank spaces and numbers, *s* = location of numbers to add together, *H* responds with that sum mod 10
- However, grid has to have 1000 digits, *H* searches for 19 of them

Previous work – Hopper (2)

- Learning parity with noise: s = n-bit vector, challenge is c = n-bit vector, response is $c \bullet s$
- Authenticate of correct exactly k out of m times – introduces noise factor (errors), essentially makes it hard to find a good data set to work with
- LPN is NP-Hard

Previous work – Hopper (3)

- Came up with methods for ensuring randomness in incorrect answers
- Also came up with ways to detect invalid challenges – all challenges must satisfy certain properties, or else *H* responds randomly, this prevents active *A* from creating arbitrary challenges

Previous work – Li (1)

- Presents two components of a secure authentication scheme
  - “Twins” – essentially LPN with superfluous noise
  - “Foxtail” – feed the unique response into a many-to-one function, so that the attacker doesn’t know what the actual response is. No evidence this is any more secure

Previous work – Li (2)

- Very elaborate demo scheme on the web, combines several schemes (mostly modified Déjà Vu and foxtailing)
- Takes a very long time to authenticate, especially with the recommended security parameters

Linear congruency

- Based on Manuel's scheme, uses mapping from letters $\rightarrow$ digits
- Challenge is English word (or string) = $a_1a_2\ldots a_n$
- x = sum of all letters in challenge
- $b_1 = n$, $b_i = r_{i-1}$
- $r_i = a_i * x + b_i$
- Response = $r_1||r_2||\ldots||r_n$
- Averages 40 seconds per response

Grid / matrix

- Large grid of letters, each one maps to a digit via previously mentioned letter mapping
- *s* = location of digits to add together, or rules to find locations that depend on content of grid
- *H* responds with sum mod 10, response time a lot faster (20 x 20 grid ~ 4 seconds)

Visual map

- Similar to Matsumoto’s map scheme
- Numbers (letters) at certain locations in mock city layout are added together mod 10
- Can (like most visual schemes) be made to detect if *C* or *A* generated the challenge
- Easy to memorize and execute, but can these locations be the same each time?

Two subsets (Avrim)

- Challenge is n-bit vector, s = two subsets of size $\log(n)$ (based on position in vector)
- r_1 = mode of first subset
- r_2 = sum of second subset
- Response = $r_1 \oplus r_2$
- Can be given over the phone – no terminal required
- Problem: generalize to mod 10 or 100

Possible attacks

- Replaying same attack to *H* multiple times to defeat LPN
- Using differential cryptanalysis techniques – must have detection of invalid challenges
- Man-in-the-middle attack – very hard to solve

Questions (1)

- How many bits per response?
  - Security vs. response time
  - How sure do we want to be? Probability of *A* succeeding on random guesses is generally exponentially small
  - Number of responses required depends on range of responses: ASCII char ~ 6.5 bits, digit ~ 3.3 bits

Questions (2)

- How long can it take?
  - HCI issue – what are people willing to do to increase security?
- How much can people remember?
  - What if they choose the secret? Can they be trained to remember more?
- How quickly and accurately can people do math?
  - Psychological limits of ~90% of people

Questions (3)

- What are people good at doing?
  - Visual recognition is easy
  - AES_K seems to be hard…
- How big should *s* be?
  - How much is needed to guarantee *H* is who he says he is?
  - How often should *s* change?

Current considerations

- Focusing on problems that machine learning people find to be difficult (Avrim)
- LPN is NP-Hard, should that be a requirement?
- Combining different schemes

Current directions

- Design of new protocols and combining existing ones to increase security and human executability
  - Visual map + two subsets with LPN
- Proving security bounds on response functions
- Ways to make it easy for people to make letter mappings

Any questions?