CSE 5095 & ECE 6095 – Spring 2016 – Instructor Marten van Dijk

Lecture Your System Security Topic #

# User Authentication

**Hoda Maleki**
Department of Electrical & Computer Engineering
University of Connecticut
Email: hoda.maleki@engr.uconn.edu

Based on and extracted from Nickolai Zeldovitch, Computer System Security, course material at http://css.csail.mit.edu/6.858/2014/
And paper: “Honeywords: Making Password-Cracking Detectable” by Juels, Rivest, 2013 and paper: "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes“ by Bonneau, Joseph, et al., SP, IEEE, 2012.
With help from Marten van Dijk

## Outline

- Definition of Authentication
- User Authentication factors
  - Password
  - Evaluation Factors
  - Compare Different User Authentication Factors
  - Multi Factor Authentication
- Honeyword
  - Motivation
  - Definition of Honeyword
  - Honeyword Generation Methods

## User Authentication

- User Authentication: Is the process of verifying the identity of a person who wants to access the system by comparing the information given by the user and what is stored in the system's database.

## Authentication factors

- Something user knows (i.e. password)
- Something user has (i.e. smart card)
- Something user is (i.e. biometric information such as fingerprint)

## Something you know

- Is the most common authentication factor, which is also the easiest to attack and beat.
- Examples: Password, PIN (Personal Identification Number)
- Methods of storage:
  - Clear text
  - Hashed password
  - Hashed with sault
  - others
- Suggestion: select long password (at least 15 character) which you are able to remember and be combination of capital letter, small letter, number, and special characters.

## Something you possess

- Is an item such as smart card or token with embedded certificate that is used to identify the user.
- Smart card: Contains credentials which certifies it's owner when is inserted into the reader.
- Token: Is a device that each time generates a number. This number is synchronized with an authentication server.
- It is better to combine it with other authentication methods.
  - E.g. smart card with PIN
  - E.g. token with user and password

## Something you are

- Biometric methods such as fingerprint, iris scan provide factors that the user is for authentication.
- Most common biometric method is fingerprint. Laptops and cellphones mostly have fingerprint readers.

## Password (Something you know)

- Currently, username/password is mostly used for user authentication.
- Passwords
  - Low entropy (how to calculated entropy?)
  - Back-up security questions with low entropy
  - A single password is often used for multiple sites.
- Is there any authentication scheme which totally dominates passwords?
  - In order to answer this question we have to have a look at the password scheme and how it works,
  - Desirable properties for authentication scheme
  - Compare other schemes with password

# How password works?

- What is password?
  - A secret shared between the user and the server.
- Implementation:
  - Save in clear text
    - Issue: passwords are in cleartext. By compromising the server, adversary can recover all user/password pairs.
  - Use hash functions in order to store the password
    - How it works? User sends the password in clear text, server applies the hash function and does a table look up.
    - Issue: adversary may apply dictionary attack.
      - Reason: top 5000 password values cover 20% of users.
        - attacker is able to optimize the hash function to be applied faster (i.e. zero-based optimization)

## How password works?

- Use key-derivation function, such as PBKDF2 or Bcrypt.
  - More calculation needs to be done, can be slow.
  - Issue: build rainbow table
- Use salt to hash a password.
  - Salt: additional random input that is added to the password before applying hash function.
    - Stores in plaintext.
    - Why better? Adversary cannot use a single rainbow table, since different salts are added to the same password that results in different hashed value.
    - Strength: long salts, change salt whenever password is changed by the user.

## Password transition

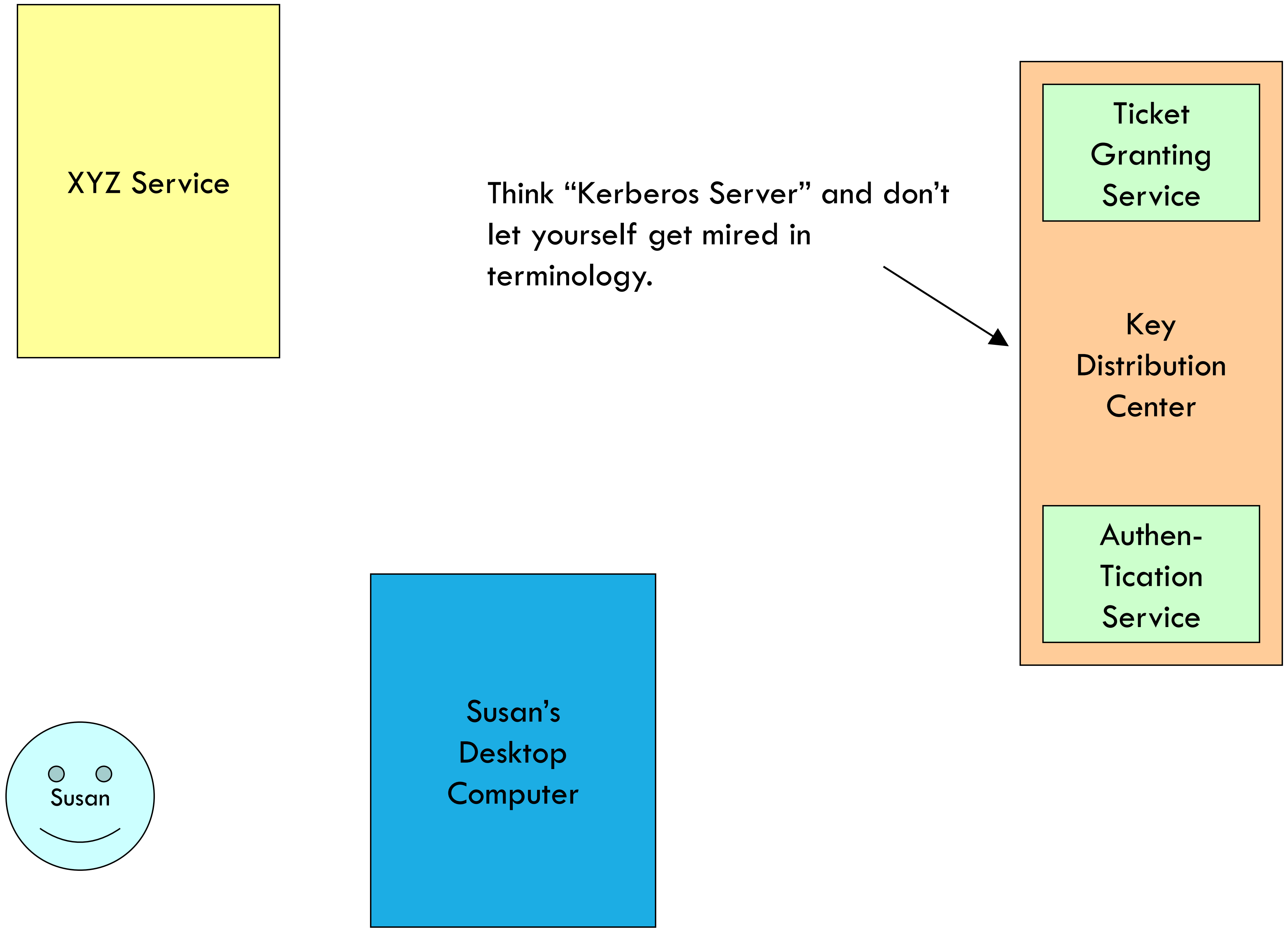
- Different solutions, such as send in plain text, use encryption, hashed password.
  - Issue: man-in –the-middle attack (server doesn't authenticate itself), replay attack.
  - Solution: use challenge-response protocol. Send the hash of the challenge with the password. This prevents MITM and replay attack.
    - Does it prevent brute-force attack, if the server is the adversary?
      - Use expensive hash, using salt OR user also uses random number as nonce (against rainbow table).
- Secure Remote Password Protocol
  - Discrete logarithm problem
  - Anti-hammering defense, i.e., disconnect and not allow the user to connect for certain amount of time if the user enters invalid password for 3 times, to prevent brute-force attack .

## Example of weak authentication system

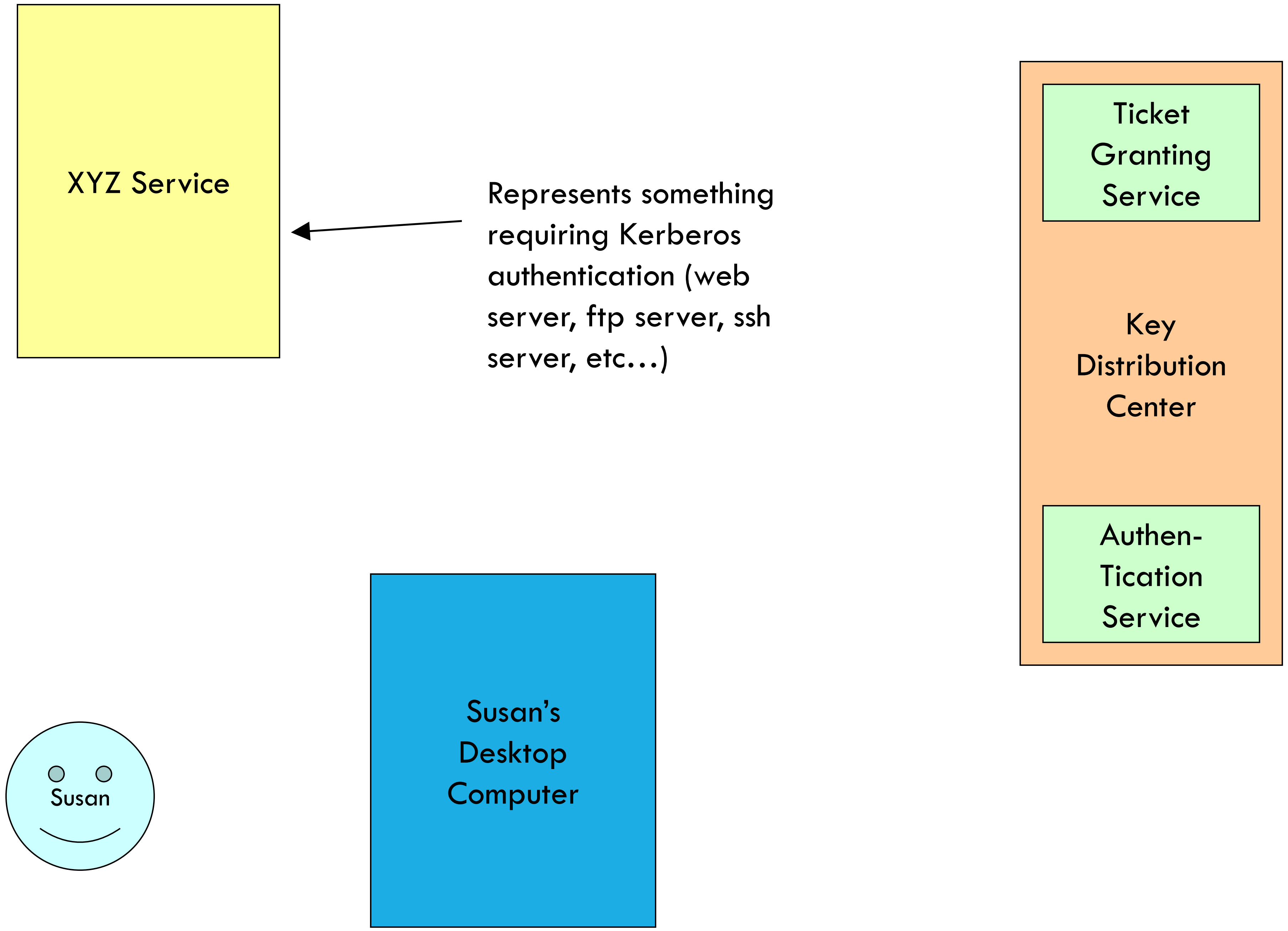
- Kerberos 4,5
  - How Kerberos works?
    - To obtain services from an application server in a “Kerberized" environment:
      - A client must First obtain a Kerberos ticket from the authentication server. This ticket contains, among other things, a section of data encrypted with the secret key belonging to the requested service.
      - The client presents this ticket to the application server, which can then verify the ticket for authenticity. Since the client does not know the server's secret key, it cannot forge a valid ticket, nor can it tamper with the contents of a ticket without being detected.
    - The actual procedure for obtaining, storing, and using tickets is divided into two steps:
      - When the user first logs in and enters his password, the client software uses the password to obtain a special ticket known as a TicketGranting Ticket (TGT) from the central authentication server.
      - When a user requires access to a Kerberized service, the client software presents the TGT to the Ticket-Granting Server (TGS), which then issues a ticket for that particular service. This servicespecic ticket is then used to authenticate the actual requests for service.

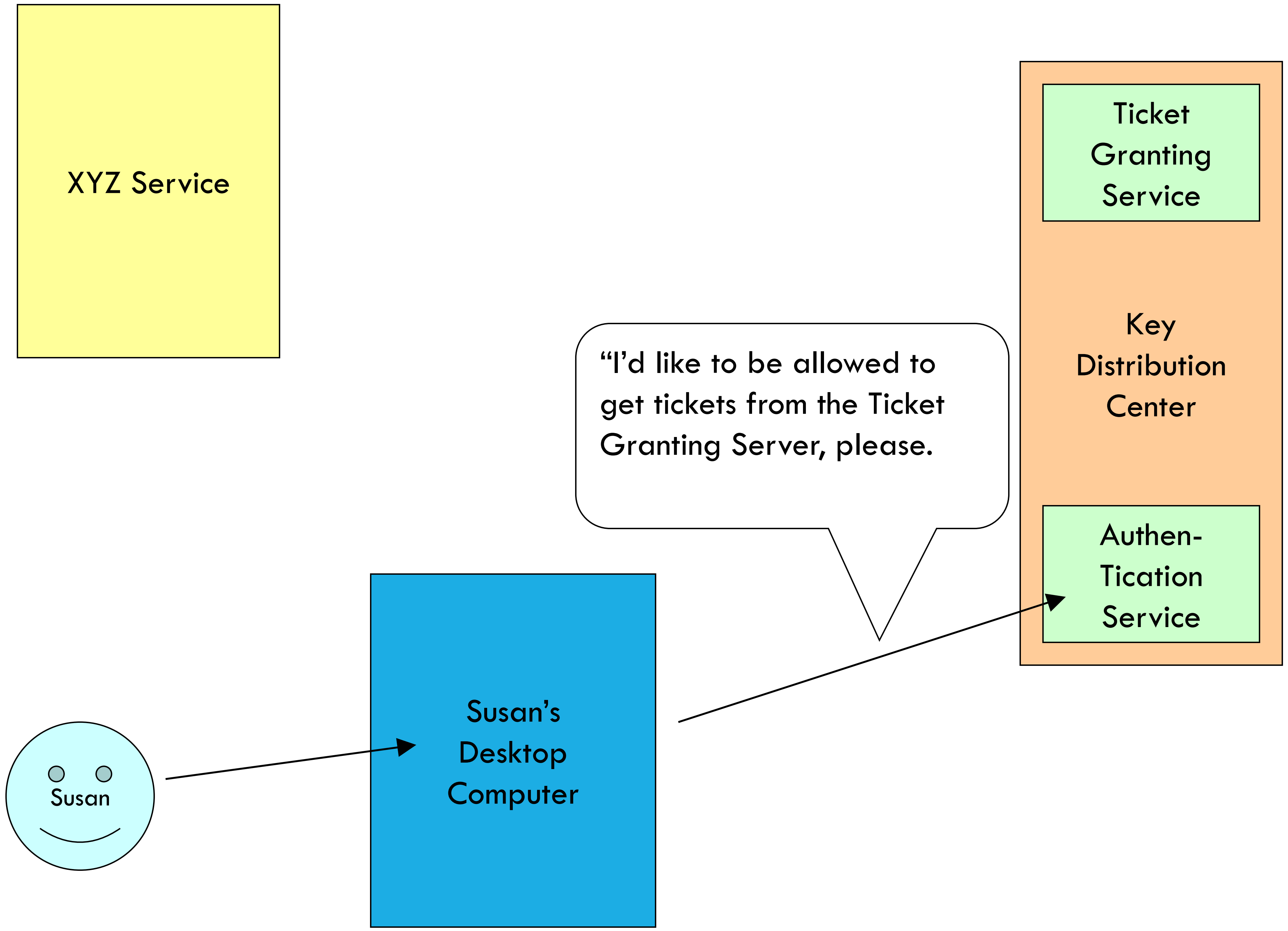
Once a user has a valid TGT, the application software can automatically obtain service-specific tickets without human intervention.

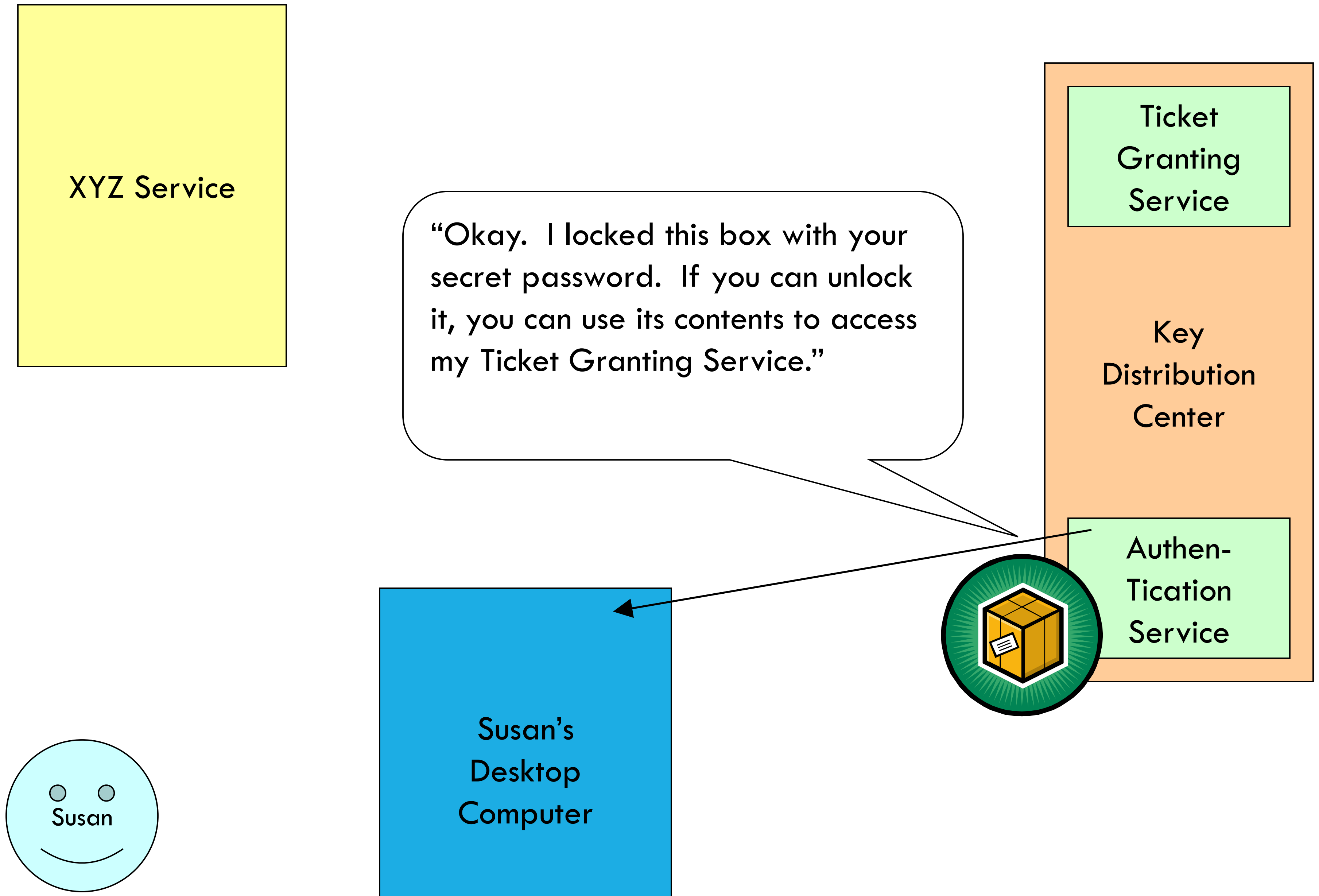
Wu, Thomas D. "A Real-World Analysis of Kerberos Password Security." NDSS. 1999.

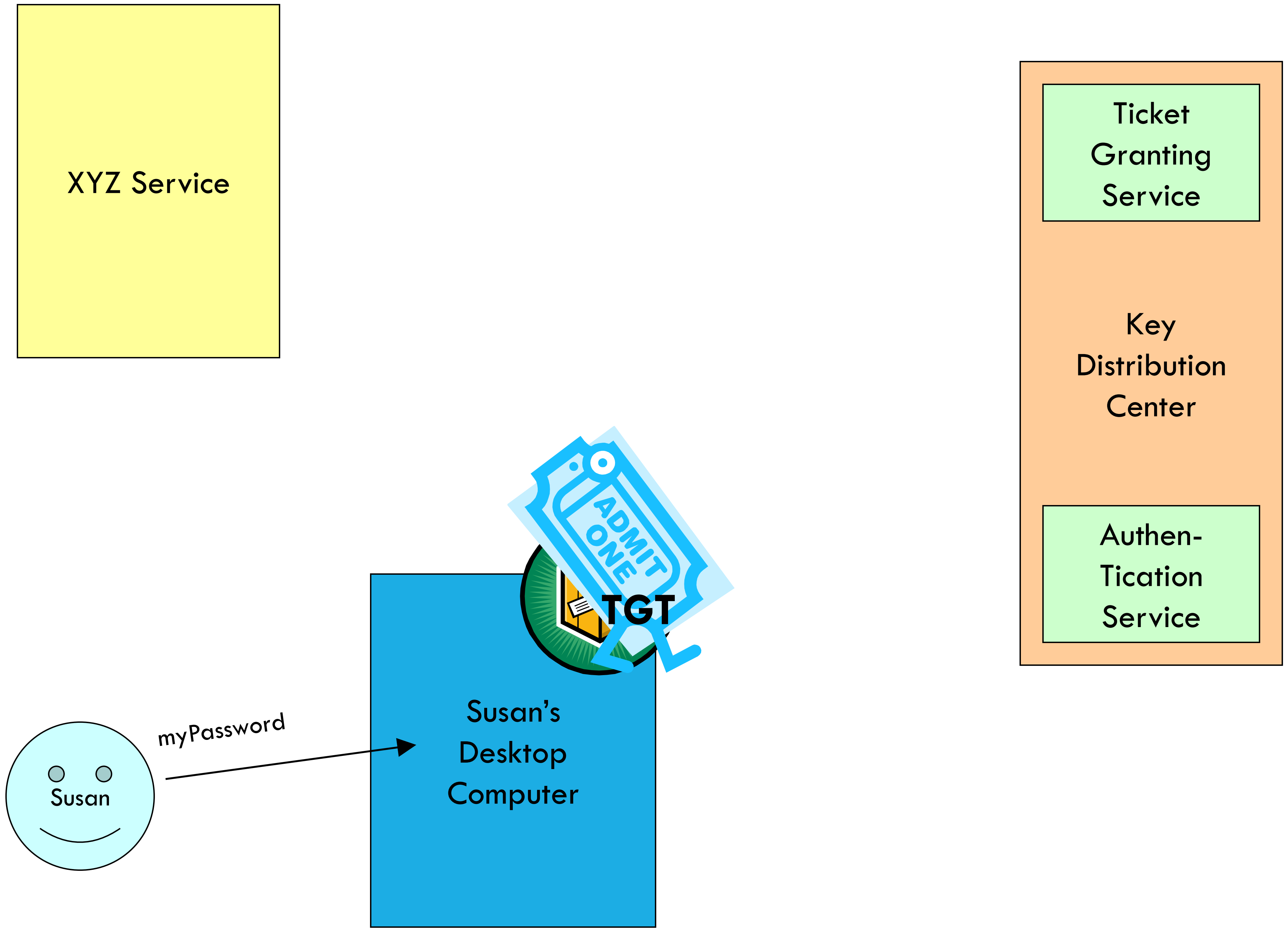


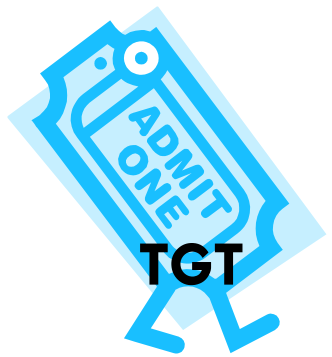
Think "Kerberos Server" and don't let yourself get mired in terminology.


XYZ Service
Represents something requiring Kerberos authentication (web server, ftp server, ssh server, etc…)
Ticket Granting Service
Key Distribution Center
Authen-Tication Service
Susan's Desktop Computer
Susan


XYZ Service
Ticket Granting Service
Key Distribution Center
Authen-Tication Service
“I’d like to be allowed to get tickets from the Ticket Granting Server, please.
Susan’s Desktop Computer
Susan


XYZ Service
Ticket Granting Service
Key Distribution Center
Authen-Tication Service
"Okay. I locked this box with your secret password. If you can unlock it, you can use its contents to access my Ticket Granting Service."
Susan's Desktop Computer
Susan

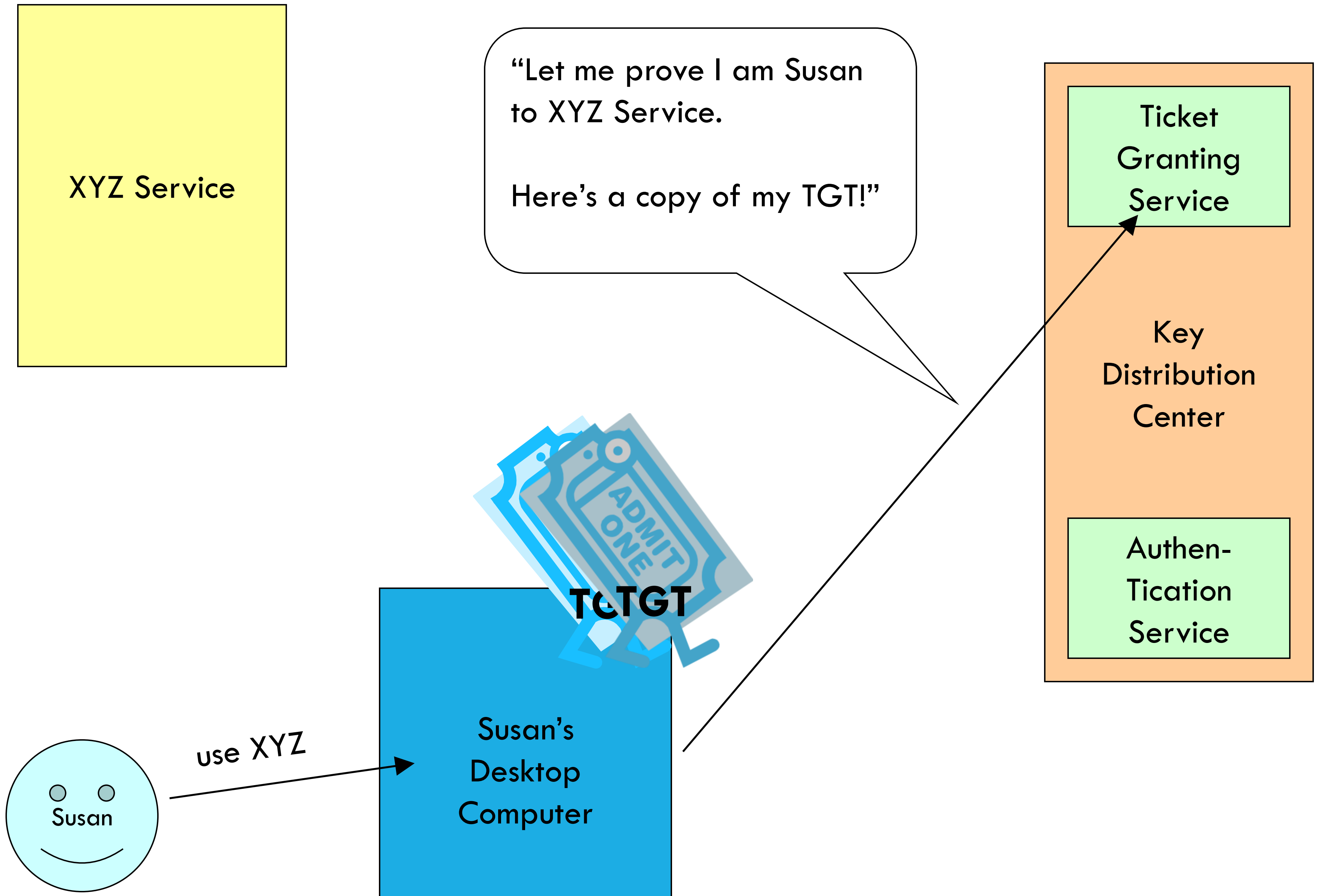

XYZ Service
Ticket Granting Service
Key Distribution Center
Authen-Tication Service
ADMIT ONE
TGT
myPassword
Susan
Susan's Desktop Computer

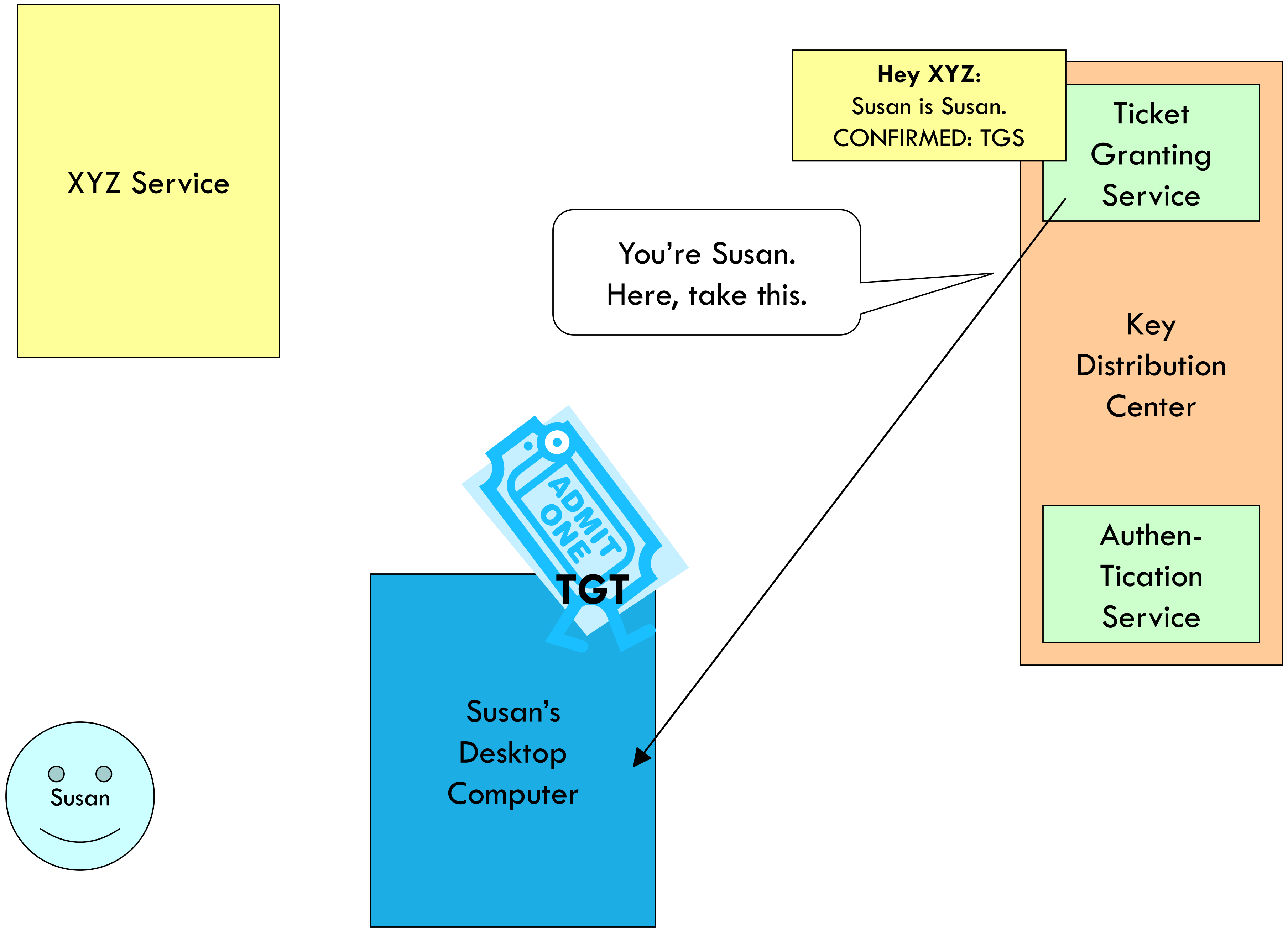


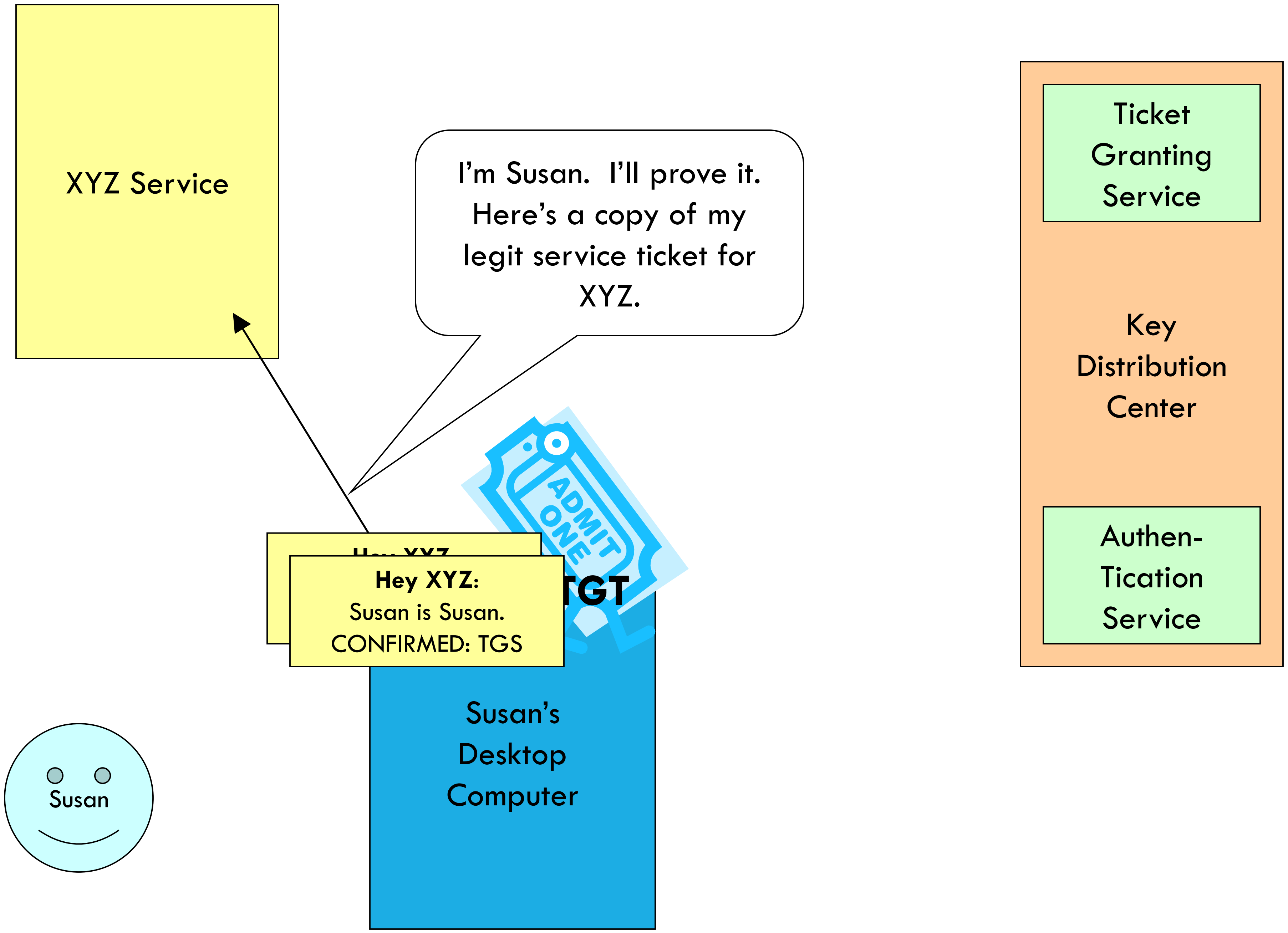
Because Susan was able to open the box (decrypt a message) from the Authentication Service, she is now the owner of a shiny “Ticket-Granting Ticket”.

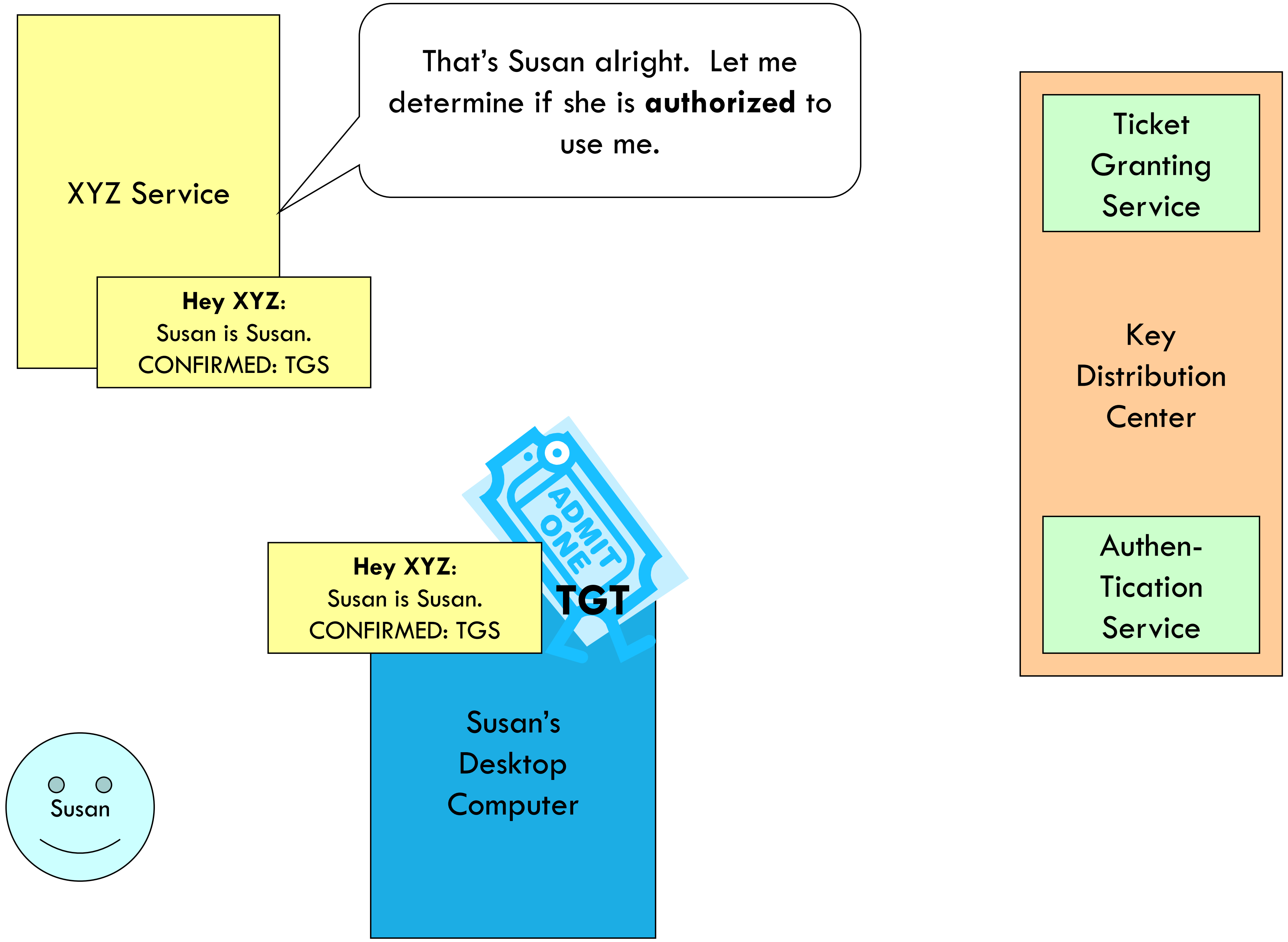
The Ticket-Granting Ticket (TGT) must be presented to the Ticket Granting Service in order to acquire “service tickets” for use with services requiring Kerberos authentication.

The TGT contains no password information.


XYZ Service
"Let me prove I am Susan to XYZ Service.
Here's a copy of my TGT!"
Ticket Granting Service
Key Distribution Center
Authen-Tication Service
ADMIT ONE
TGTGT
Susan's Desktop Computer
use XYZ
Susan


XYZ Service
Hey XYZ: Susan is Susan. CONFIRMED: TGS
Ticket Granting Service
You're Susan. Here, take this.
Key Distribution Center
ADMIT ONE
TGT
Authen-Tication Service
Susan's Desktop Computer
Susan


XYZ Service
I'm Susan. I'll prove it. Here's a copy of my legit service ticket for XYZ.
Ticket Granting Service
Key Distribution Center
Authen-Tication Service
ADMIT ONE
TGT
Hey XYZ
Hey XYZ:
Susan is Susan.
CONFIRMED: TGS
Susan's Desktop Computer
Susan


XYZ Service
That's Susan alright. Let me determine if she is **authorized** to use me.
Ticket Granting Service
Key Distribution Center
Authen-Tication Service
**Hey XYZ:** Susan is Susan. CONFIRMED: TGS
ADMIT ONE
TGT
**Hey XYZ:** Susan is Susan. CONFIRMED: TGS
Susan's Desktop Computer
Susan

## Kerberos vulnerability

- Normally, if the user enters an incorrect password, the initial decryption attempt produces a gibberish packet, which causes the Kerberos client software to notify the user and discard the packet.
- But
  - what if the client software, instead of throwing away the packet after each attempt, allowed the user to try decrypting the same packet again with different passwords?
  - what if, instead of having a human typing in different passwords, the software automated the procedure, pulling in passwords from a dictionary as fast as it could check them?
- Since the TGT has a fixed, publicly-known format, the software could determine if it had found the correct password by looking for one that decrypted the TGT properly (dictionary attack).

Wu, Thomas D. "A Real-World Analysis of Kerberos Password Security." NDSS. 1999.

# Password Recovery

- Since the recovery questions can be used to reset passwords, the strength of authentication scheme is Min(password-entropy, recovery-question-entropy)
  - Issue: the answer to most of recovery question is easy to guess, specially if the adversary can get more information about the victim via social media profiles.
    - Ex: what is your favorite color?
    - Ex: what is your favorite movie?

## Evaluation factors for authentication scheme

- Goal: to evaluate different authentication scheme, specially passwords.
- In Paper: “The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes”
  - To evaluate all different authentication scheme, a standard benchmark and framework is introduced with 25 properties for analyzing wide spectrum of benefits they offer, when compared to text passwords.
  - To rate the pros and cons of each scheme, this framework is used extensively on 35 password replacement schemes.
  - Main focus in the rating process is user authentication on the web, specifically from client devices like PCs to remote verifiers. That means, human-to-machine authentication, but not machine-to-machine.
- The benefit of each scheme to be considered are placed under three categories:
  - Usability: Ease of interacting with the user authentication scheme.
  - Deployability: The ease of combine the authentication scheme in systems.
  - Security: The attack types that the scheme prevents.

## Usability Benefits

1. Memorywise Effortless: Users of the scheme do not have to remember any secrets at all.
   - Quasi-Memorywise effortless: if users have to remember one secret for everything
2. Scalable for users
   - Using the same scheme for hundreds of accounts does not increase burden on the user.
   - By Scalable, it means from user's cognitive load perspective , but not system resource perspective
3. Nothing-to-Carry: Users do not need to carry an additional physical object such as piece of paper, electronic device, mechanical key
   - Quasi-Nothing-to-Carry: is awarded if the object is one that they'd carry everywhere all the time anyway, such as their mobile phone, but not if it's their computer (including tablets).
4. Physically-Effortless: The authentication process does not require physical (as opposed to cognitive) user effort beyond, say, pressing a button.
   - Quasi-Physically-Effortless: If the user's effort is limited to speaking.

## Usability Benefits

5. Easy-to-Learn: Users who don't know the scheme can figure it out and learn it without too much trouble, and then easily recall how to use it.
6. Efficient-to-Use: The time the user must spend for each authentication is acceptably short. The time required for setting up a new association with a verifier, although possibly longer than that for authentication, is also reasonable.
7. Infrequent-Errors: The task that users must perform to log in usually succeeds when performed by a legitimate and honest user. In other words, the scheme isn't so hard to use or unreliable that genuine users are routinely rejected.
8. Easy-Recovery-from-Loss: A user can conveniently regain the ability to authenticate if the token is lost or the credentials forgotten. This combines usability aspects such as: low latency before restored ability; low user inconvenience in recovery (e.g., no requirement for physically standing in line); and assurance that recovery will be possible, for example via built-in backups or secondary recovery schemes. If recovery requires some form of reenrollment, this benefit rates its convenience.

## Deployability Benefits

1. Accessible: Users who can use passwords are not prevented from using the scheme by disabilities or other physical (not cognitive) conditions.
2. Negligible-Cost-per-User: The total cost per user of the scheme, adding up the costs at both the prover's end (any devices required) and the verifier's end (any share of the equipment and software required), is negligible. The scheme is plausible for startups with no per-user revenue.
3. Server-Compatible: At the verifier's end, the scheme is compatible with text-based passwords. Providers don't have to change their existing authentication setup to support the scheme.
4. Browser-Compatible: Users don't have to change their client to support the scheme and can expect the scheme to work when using other machines with an up-to-date, standards-compliant web browser and no additional software. Schemes fail to provide this benefit if they require the installation of plugins or any kind of software whose installation requires administrative rights.
   - Quasi-Browser-Compatible: If they rely on non-standard but very common plugins, e.g., Flash.
5. Mature: The scheme has been implemented and deployed on a large scale for actual authentication purposes beyond research. Indicators to consider for granting the full benefit may also include whether the scheme has undergone user testing, whether the standards community has published related documents, whether open-source projects implementing the scheme exist, whether anyone other than the implementers has adopted the scheme, the amount of literature on the scheme and so forth.
6. Non-Proprietary: Anyone can implement or use the scheme for any purpose without having to pay royalties to anyone else. The relevant techniques are generally known, published openly and not protected by patents or trade secrets.

## Security Benefits

1. Resilient-to-Physical-Observation: An attacker cannot impersonate a user after observing them authenticate one or more times.
   - Quasi-Resilient-to-Physical-Observation: if the scheme could be broken only by repeating the observation more than, say, 10–20 times. Attacks include shoulder surfing, filming the keyboard, recording keystroke sounds, or thermal imaging of keypad.
2. Resilient-to-Targeted-Impersonation: It is not possible for an acquaintance (or skilled investigator) to impersonate a specific user by exploiting knowledge of personal details (birth date, names of relatives etc.). Personal knowledge questions are the canonical scheme that fails on this point.
3. Resilient-to-Throttled-Guessing: An attacker whose rate of guessing is constrained by the verifier cannot successfully guess the secrets of a significant fraction of users. The verifier-imposed constraint might be enforced by an online server, a tamper-resistant chip or any other mechanism capable of throttling repeated requests.
4. Resilient-to-Unthrottled-Guessing: An attacker whose rate of guessing is constrained only by available computing resources cannot successfully guess the secrets of a significant fraction of users. We might for example grant this benefit if an attacker capable of attempting up to $2^{40}$ or even $2^{64}$ guesses per account could still only reach fewer than 1% of accounts. Lack of this benefit is meant to penalize schemes where the space of credentials is not large enough to withstand brute force search (including dictionary attacks, rainbow tables and related brute force methods smarter than raw exhaustive search, if credentials are user-chosen secrets).

# Security Benefits

5. Resilient-to-Internal-Observation: An attacker cannot impersonate a user by intercepting the user's input from inside the user's device (e.g., by keylogging malware) or eavesdropping on the cleartext communication between prover and verifier.
   - Quasi-Resilient-to-Internal-Observation: If the scheme could be broken only by intercepting input or eavesdropping cleartext more than certain amount. This penalizes schemes that are not replay-resistant, whether because they send a static response or because their dynamic response countermeasure can be cracked with a few observations.

6. Resilient-to-Phishing: An attacker who simulates a valid verifier, cannot collect credentials that can later be used to impersonate the user to the actual verifier. This penalizes schemes allowing phishers to get victims to authenticate to lookalike sites and later use the harvested credentials against the genuine sites.

7. Resilient-to-Theft: If the scheme uses a physical object for authentication, the object cannot be used for authentication by another person who gains possession of it.
   - Quasi-Resilient-to-Theft: if the protection is achieved with the modest strength of a PIN, even if attempts are not rate controlled, because the attack doesn't easily scale to many victims.

8. No-Trusted-Third-Party: The scheme does not rely on a trusted third party (other than the prover and the verifier) who could, upon being attacked or otherwise becoming untrustworthy, compromise the prover's security or privacy.
   - This property makes an important point: a lot of authentication problems would become easier if we could just trust one party to store passwords, run the password servers, etc. However, single points of failure are bad, since attackers can focus all of their energy on that point.

9. Requiring-Explicit-Consent: The authentication process cannot be started without the explicit consent of the user. This is both a security and a privacy feature (a rogue wireless RFID-based credit card reader embedded in a sofa might charge a card without user knowledge or consent).

10. Unlinkable: Colluding verifiers cannot determine, from the authenticator alone, whether the same user is authenticating to both. This is a privacy feature.

## Authentication Schemes

- Several authentication schemes are invented as replacement to passwords. The following are considered in the paper:
  - Password management software
  - Federal login protocols
  - Graphical Password schemes
  - Cognitive authentication schemes
  - Hardware tokens
  - One-time passwords
  - Phone aided schemes
- Thus schemes use the methods explained in order to strength the password against different adversary capabilities

## Evaluation of scheme with ratings

- Evaluation of Legacy Passwords
  - Highly scores in Deployability
.

| Survey details | Advantages | Disadvantages |
|---|---|---|
| -3 decades ago, the researchers were able to guess over 75% of users' passwords.<br>-Corporate password users tend to copy the passwords on post-it notes.<br>- most users have many accounts for which they have forgotten their passwords.<br>-On average 25 accounts and 6 unique passwords per user. | -Nothing-to-Carry<br>-Easy-to-Learn: The key reason why password schemes are so popular.<br>-Efficient-to-Use (as most users type only a few characters)<br>- Quasi-Infrequent-Errors( because of typos): is an important reason why users pick easy-to-guess passwords.<br>- Easy-Recovery-from-Loss: They are easy to reset.<br>-Accessible<br>- Negligible-Cost-per-User<br>- Resilient-to-Theft<br>-No-Trusted-Third-Party | - Not Memorywise-Effortless<br>-Not Scalable-for-users ( must be remembered and chosen for each site)<br>-Not Resilient-to-Physical-Observation: passwords fail this test, since, e.g., they can be captured by filming the keyboard or recording keystroke sounds.<br>-Quasi-Resilient-to-Targeted-Impersonation: The authors say that passwords have is property because they could not find any studies saying that your friends or acquaintances can easily guess your password.<br>-Resilient-to-Throttled-Guessing: passwords fail because they have low entropy and skewed distribution.<br>-Resilient-to-Unthrottled-Guessing: passwords fail because they have low entropy and skewed distribution.<br>-Resilient-to-Internal-Observation: Passwords fail because they are static tokens, once you have one, you can use it until it expires or is revoked.<br>-Resilient-to-Phishing: phishing attack are very common. |

## Evaluation of scheme with ratings

- Evaluation of Encrypted Password Managers : Mozialla Firefox
  - Highly scores in Deployability.

| Survey details | Advantages | Disadvantages |
|---|---|---|
| -Automatically offers to remember passwords, optionally encrypted with master password.<br>- It pre-fills username and password when the user revisits the same web site.<br>-With its SYNC facility, the passwords can be stored, encrypted in the cloud.<br>- No typing required, except the master password once per session. | -Quasi-Memorywise-Effortless<br>-Scalable-for-Users<br>-Quasi-Nothing-to-Carry(at least to carry a smart phone)<br>- Quasi-Physically-Effortless<br>- Easy-to-Learn<br>-Efficient-to-Use<br>-Infrequent-Errors(hardly any)<br>-Quasi-Resilient-to-Targeted-Impersonation.<br>-Resilient-to-Theft<br>- Unlinkable | -Not Easy-Recovery-from-Loss ( catastrophic to lose the master password)<br>- Not Resilient-to-Throttled-Guessing<br>- Not Resilient-to-Unthrottled-Guessing |

## Evaluation of scheme with ratings

- Evaluation of Proxy Based : URRSA

| Survey details | Advantages | Disadvantages |
|---|---|---|
| -places a man in the middle between the user's machine and the server( to enable secure logins despite malware)<br>- The User password is encrypted at the proxy with 30 different keys.<br>-The codes are generated at the proxy by using 30 keys and password.<br>- User carries codes and uses at login time.<br>-The proxy never authenticates the user, but merely decrypts with agreed-upon key. | -Memorywise-Effortless<br>-Quasi-Infrequent-Errors<br>- Quasi-Server-Compatible<br>- Browser-Compatible<br>- Quasi-Resilient-to-Targeted-Impersonation.<br>- Quasi-Resilient-to-Internal-Observation<br>- Negligible-Cost-per-User | -Not Scalable-for-Users<br>-Not Nothing-to-Carry<br>- Not Physically-Effortless<br>-Not Efficient-to-Use<br>- Not Easy-Recovery-from-Loss(since no passwords are stored at the proxy)<br>- Not Mature<br>- Not Proprietary |

# Evaluation of scheme with ratings

- Evaluation of Federated Singel Sign-On : OpenID
  - Favorable from deployment point of view.

| Survey details | Advantages | Disadvantages |
|---|---|---|
| -Enables web sites to authenticate a user by redirecting to a trusted identity server.<br>-Eliminates problem of remembering different passwords for different sites.<br>- Still uses text passwords to authenticate users. | -Quasi-Memorywise-Effortless(need to remmber one master password)<br>-Scalable-for-Users (can work for multiple sites)<br>- Nothing-to-Carry<br>- Efficient-to-Use<br>- Infrequent-Errors<br>- Easy-Recovery-from-Loss( same as password reset) | - not Server-Compatible<br>- Not Resilient-to-Internal-Observation (malware can steal identity from cached cookie)<br>- Not Resilient-to-Phishing<br>-Not Unlinkable |

## Evaluation of scheme with ratings

- Evaluation of Graphical Passwords : Persuasive Cued Clickpoints (PCCP)

| Survey details | Advantages | Disadvantages |
|---|---|---|
| -Leverage natural human ability to remember images, which is believed to exceed memory for text.<br>- User is given five images to select one point on each, determining the next image displayed. | -Easy-to-Learn(usage and mental models match web passwords)<br>- Quasi-Efficient-to-Use(login times on the order of 5s to 20s exceed text passwords)<br>- Quasi-Infrequent-Errors<br>- Browser-Compatible. | - Not Memorywise-Effortless<br>-Not Scalable-for-Users<br>- Not Accessible ( for blind users)<br>- Not Server-Compatible<br>-Not Mature<br>- Not Resilient-to-Physical-Observation. |

# Evaluation of scheme with ratings

- Evaluation of Cognitive Authentication: GrIDsure

| Survey details | Advantages | Disadvantages |
|---|---|---|
| -Challenge-Response schemes that attempt to address the reply attack on passwords by having the user deliver proof that he knows the secret.<br>-If memorization and computation were no barrier then the server might challenge the user to return a cryptographic hash of the user's secret combined with a server-selected nonce.<br>- However, it is unclear if a scheme within the means of human memory and calculating ability is achievable. | - Quasi-Efficient-to-Use(unlike passwords)<br>- Negligible-Cost-per-User (in terms of technology)<br>- Browser-Compatible<br>- Resilient-to-Targeted-Impersonation | - Not Accessible.<br>- Not Server-Compatible.<br>- Not Resilient-to-Physical-Observation. |

## Evaluation of scheme with ratings

- Evaluation of Paper Tokens : OTPW

| Survey details | Advantages | Disadvantages |
|---|---|---|
| -Using paper to store long secrets in the cheapest form of a physical login token.<br>- The concept is related to military codebooks used throught history.<br>- User carries the hash pre-images, printed as 8-character values. | - Easy-Recovery-from-Loss<br>-Negligible-Cost-per-User<br>-Browser-Compatible | - Not Memorywise-Effortless.<br>- Not Scalable-for-Users<br>-Not Nothing-to-Carry( because of paper tokens)<br>-Not Physically-Effortless<br>- Not Resilient-to-Physical-Observation. |

# Evaluation of scheme with ratings

- Evaluation of Hardware tokens : RSA Secure ID

| Survey details | Advantages | Disadvantages |
|---|---|---|
| - store secrets in a dedicated tamper-resistant module.<br>- Each instance of the devide holds a secret "seed" known to the back-end.<br>- Generates a new 6 digit code from this secret every 60 seconds.<br>- User enters PIN along with this generated code.<br>- concatenation of this 4 digit PIN and the dynamic 6 digit code is called PASSCODE. | -Easy-to-Learn<br>- Quasi-Efficient-to-Use<br>-Quasi-Infrequent-Errors.(like passwords) | - Not Memorywise-Effortless<br>- Not Scalable-for-Users (needs new Token and PIN per user)<br>- Not Physically-Effortless<br>- Not Easy-Recovery-from-Loss<br>- Not Accessible (for blind users) |

# Evaluation of scheme with ratings

- Evaluation of Hardware tokens : CAP reader

| Survey details | Advantages | Disadvantages |
|---|---|---|
| - CAP reader was designed by Mastercard to protect online banking transactions.<br>-The user must put the credit card into the CAP reader and enter the PIN. Next, the reader talks to the card's embedded processor, outputs an 8-digit code which the user supplies to the web site.<br>-In practice, deployability and usability are often more important than the security, this is why CAP reader haven't taken over the world.<br>-Migration costs such as coding, debugging effort, and user training, make developers nervous.<br>-The less usable a scheme is, the more that users will complain (and try to pick easier authentication tokens that are more vulnerable to attacks.<br>- | -Easy-to-Learn<br>-Quasi-Efficient-to-Use<br>-Quasi-Infrequent-Errors.(like passwords) | - Not Memorywise-Effortless<br>- Not Scalable-for-Users (needs new Token and PIN per user)<br>- Not Physically-Effortless<br>- Not Easy-Recovery-from-Loss<br>- Not Accessible (for blind users) |

# Evaluation of scheme with ratings

- Evaluation of Mobile Phone-based : Phoolproof

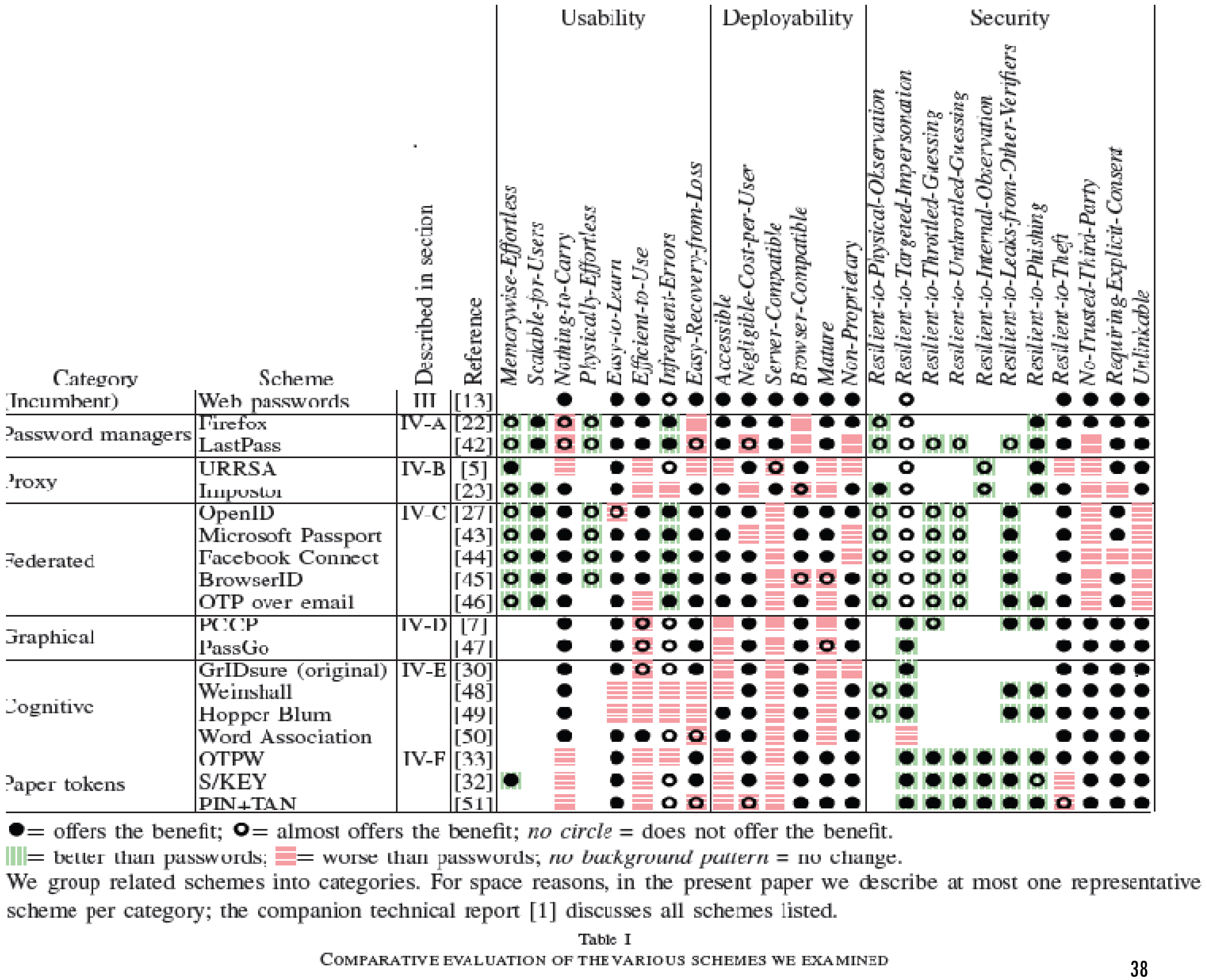
| Survey details | Advantages | Disadvantages |
|---|---|---|
| - Token is a mobile phone with special code and crypto keys.<br>- It uses public key cryptography and SSL like authentication protocol. | - Quasi-Infrequent-Errors.(like passwords)<br>- Quasi-Negligible-Cost-per-User<br>-Resilient-to-Physical-Observation<br>-Resilient-to-Impersonation<br>--Resilient-to-Throttled-Guessing<br>- Resilient-to-Unthrottled-Guessing | - Not Memorywise-Effortless.<br>- Not Scalable-for-Users<br>-Not Easy-Recovery-from-Loss |

# Evaluation of scheme with ratings

- Evaluation of Biometrics : Fingerprint recognition
  - How Big the keyspace is?
    - Fingerprint: ~13.3 bits
    - Iris scan: ~19.9bits
    - Voice recognition: ~11.7bits
  - Bits of entropy are roughly the same as passwords.

| Survey details | Advantages | Disadvantages |
|---|---|---|
| - leverage uniqueness of physical or behavioral characteristics across individuals. | -Memorywise-Effortless<br>- Scalable-for-Users<br>- Easy-to-Learn | - not -Negligible-Cost-per-User<br>-Not browse-compatible |

# Comparisons of Various Schemes

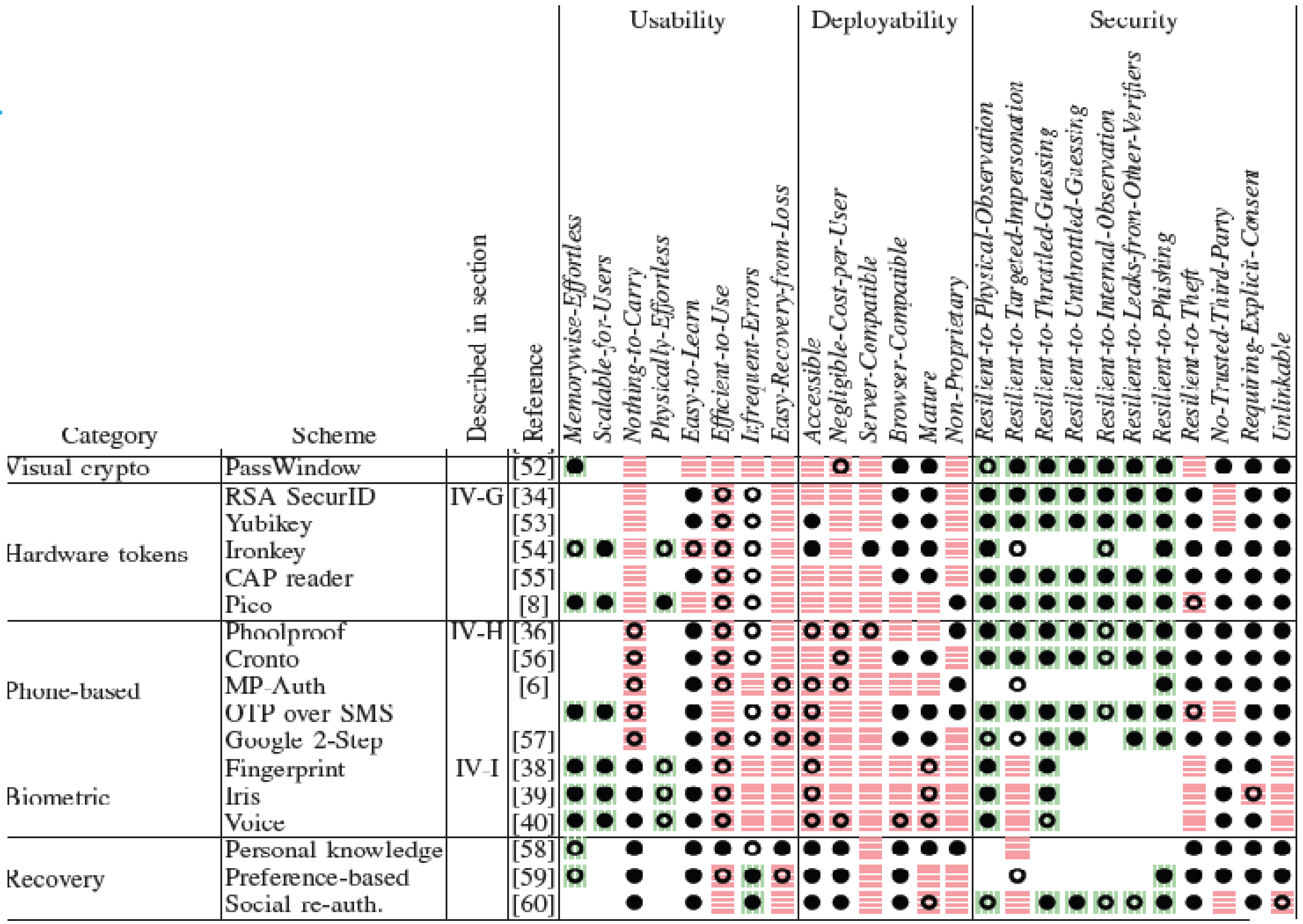


| | | | | Usability | | | | | | | | Deployability | | | | | | Security | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Category | Scheme | Described in section | Reference | *Memorywise-Effortless* | *Scalable-for-Users* | *Nothing-to-Carry* | *Physically-Effortless* | *Easy-to-Learn* | *Efficient-to-Use* | *Infrequent-Errors* | *Easy-Recovery-from-Loss* | *Accessible* | *Negligible-Cost-per-User* | *Server-Compatible* | *Browser-Compatible* | *Mature* | *Non-Proprietary* | *Resilient-to-Physical-Observation* | *Resilient-to-Targeted-Impersonation* | *Resilient-to-Throttled-Guessing* | *Resilient-to-Unthrottled-Guessing* | *Resilient-to-Internal-Observation* | *Resilient-to-Leaks-from-Other-Verifiers* | *Resilient-to-Phishing* | *Resilient-to-Theft* | *No-Trusted-Third-Party* | *Requiring-Explicit-Consent* | *Unlinkable* |
| (Incumbent) | Web passwords | III | [13] | | | ● | | ● | ● | ○ | ● | ● | ● | ● | ● | ● | ● | | ○ | | | | | | ● | ● | ● | ● |
| Password managers | Firefox | IV-A | [22] | ○🟩 | ●🟩 | ○🟥 | ○🟩 | ● | ● | ●🟩 | 🟥 | ● | ● | ● | 🟥 | ● | ● | ○🟩 | ○ | | | | | ●🟩 | ● | ● | ● | ● |
| | LastPass | | [42] | ○🟩 | ●🟩 | ○🟥 | ○🟩 | ● | ● | ●🟩 | ○🟥 | ● | ○🟥 | ● | 🟥 | ● | 🟥 | ○🟩 | ○ | ○🟩 | ○🟩 | | ○🟩 | ●🟩 | ● | 🟥 | ● | ● |
| Proxy | URRSA | IV-B | [5] | ●🟩 | | 🟥 | | ● | 🟥 | ○ | 🟥 | 🟥 | ● | ○🟥 | ● | 🟥 | 🟥 | | ○ | | | ○🟩 | | ●🟩 | 🟥 | 🟥 | ● | ● |
| | Impostor | | [23] | ○🟩 | ●🟩 | ● | | ● | 🟥 | 🟥 | ● | ● | 🟥 | ● | ○🟥 | 🟥 | ● | ●🟩 | ○ | | | ○🟩 | | ●🟩 | ● | 🟥 | 🟥 | ● |
| Federated | OpenID | IV-C | [27] | ○🟩 | ●🟩 | ● | ○🟩 | ○🟥 | ● | ●🟩 | ● | ● | ● | 🟥 | ● | ● | ● | ○🟩 | ○ | ○🟩 | ○🟩 | | ●🟩 | | ● | 🟥 | ● | 🟥 |
| | Microsoft Passport | | [43] | ○🟩 | ●🟩 | ● | ○🟩 | ● | ● | ●🟩 | ● | ● | 🟥 | 🟥 | ● | ● | 🟥 | ○🟩 | ○ | ○🟩 | ○🟩 | | ●🟩 | | ● | 🟥 | ● | 🟥 |
| | Facebook Connect | | [44] | ○🟩 | ●🟩 | ● | ○🟩 | ● | ● | ●🟩 | ● | ● | ● | 🟥 | ● | ● | 🟥 | ○🟩 | ○ | ○🟩 | ○🟩 | | ●🟩 | | ● | 🟥 | 🟥 | 🟥 |
| | BrowserID | | [45] | ○🟩 | ●🟩 | ● | ○🟩 | ● | ● | ●🟩 | ● | ● | ● | 🟥 | ○🟥 | ○🟥 | ● | ○🟩 | ○ | ○🟩 | ○🟩 | | ●🟩 | | ● | 🟥 | ● | 🟥 |
| | OTP over email | | [46] | ○🟩 | ●🟩 | ● | | ● | 🟥 | ●🟩 | ● | ● | ● | 🟥 | ● | 🟥 | ● | ○🟩 | ○ | ○🟩 | ○🟩 | | ●🟩 | ●🟩 | ● | 🟥 | ● | 🟥 |
| Graphical | PCCP | IV-D | [7] | | | ● | | ● | ○🟥 | ○ | ● | 🟥 | ● | 🟥 | ● | 🟥 | ● | | ●🟩 | ○🟩 | | | ●🟩 | ●🟩 | ● | ● | ● | ● |
| | PassGo | | [47] | | | ● | | ● | ○🟥 | ○ | ● | 🟥 | ● | 🟥 | ● | ○🟥 | ● | | ●🟩 | | | | | | ● | ● | ● | ● |
| Cognitive | GrIDsure (original) | IV-E | [30] | | | ● | | ● | ○🟥 | ○ | ● | 🟥 | ● | 🟥 | ● | 🟥 | 🟥 | | ●🟩 | | | | | | ● | ● | ● | ● |
| | Weinshall | | [48] | | | ● | | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | ● | 🟥 | ● | 🟥 | ● | ○🟩 | ●🟩 | | | | ●🟩 | ●🟩 | ● | ● | ● | ● |
| | Hopper Blum | | [49] | | | ● | | 🟥 | 🟥 | 🟥 | 🟥 | ● | ● | 🟥 | ● | 🟥 | ● | ○🟩 | ●🟩 | | | | ●🟩 | ●🟩 | ● | ● | ● | ● |
| | Word Association | | [50] | | | ● | | ● | ● | ○ | ○🟥 | ● | ● | 🟥 | ● | 🟥 | ● | | 🟥 | | | | | | ● | ● | ● | ● |
| Paper tokens | OTPW | IV-F | [33] | | | 🟥 | | ● | 🟥 | 🟥 | ● | 🟥 | ● | 🟥 | ● | ● | ● | | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ● | ● | ● | ● |
| | S/KEY | | [32] | ●🟩 | | 🟥 | | ● | 🟥 | ○ | ● | 🟥 | ● | 🟥 | ● | ● | ● | | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ○🟩 | 🟥 | ● | ● | ● |
| | PIN+TAN | | [51] | | | 🟥 | | ● | 🟥 | ○ | ○🟥 | 🟥 | ○🟥 | 🟥 | ● | ● | ● | | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ○🟥 | ● | ● | ● |

●= offers the benefit; ○= almost offers the benefit; *no circle* = does not offer the benefit.
🟩= better than passwords; 🟥= worse than passwords; *no background pattern* = no change.
We group related schemes into categories. For space reasons, in the present paper we describe at most one representative scheme per category; the companion technical report [1] discusses all schemes listed.

Table I
COMPARATIVE EVALUATION OF THE VARIOUS SCHEMES WE EXAMINED

# Comparisons of Various Schemes



| | | | | Usability | | | | | | | | Deployability | | | | | | Security | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Category | Scheme | Described in section | Reference | *Memorywise-Effortless* | *Scalable-for-Users* | *Nothing-to-Carry* | *Physically-Effortless* | *Easy-to-Learn* | *Efficient-to-Use* | *Infrequent-Errors* | *Easy-Recovery-from-Loss* | *Accessible* | *Negligible-Cost-per-User* | *Server-Compatible* | *Browser-Compatible* | *Mature* | *Non-Proprietary* | *Resilient-to-Physical-Observation* | *Resilient-to-Targeted-Impersonation* | *Resilient-to-Throttled-Guessing* | *Resilient-to-Unthrottled-Guessing* | *Resilient-to-Internal-Observation* | *Resilient-to-Leaks-from-Other-Verifiers* | *Resilient-to-Phishing* | *Resilient-to-Theft* | *No-Trusted-Third-Party* | *Requiring-Explicit-Consent* | *Unlinkable* |
| Visual crypto | PassWindow | | [52] | ●🟩 | | 🟥 | | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | ○🟥 | 🟥 | ● | ● | 🟥 | ○🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | 🟥 | ● | ● | ● |
| Hardware tokens | RSA SecurID | IV-G | [34] | | | 🟥 | | ● | ○🟥 | ○ | 🟥 | 🟥 | 🟥 | 🟥 | ● | ● | 🟥 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ● | 🟥 | ● | ● |
| | Yubikey | | [53] | | | 🟥 | | ● | ○🟥 | ○ | 🟥 | ● | 🟥 | 🟥 | ● | ● | 🟥 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ● | 🟥 | ● | ● |
| | Ironkey | | [54] | ○🟩 | ●🟩 | 🟥 | ○🟩 | ○🟥 | ○🟥 | ○ | 🟥 | ● | 🟥 | ● | ● | ● | 🟥 | ●🟩 | ○ | | | ○🟩 | | ●🟩 | ● | ● | ● | ● |
| | CAP reader | | [55] | | | 🟥 | | ● | ○🟥 | ○ | 🟥 | 🟥 | 🟥 | 🟥 | ● | ● | 🟥 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ● | ● | ● | ● |
| | Pico | | [8] | ●🟩 | ●🟩 | 🟥 | ●🟩 | 🟥 | ○🟥 | ○ | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | ● | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ○🟥 | ● | ● | ● |
| Phone-based | Phoolproof | IV-H | [36] | | | ○🟥 | | ● | ○🟥 | ○ | 🟥 | ○🟥 | ○🟥 | ○🟥 | 🟥 | 🟥 | ● | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ○🟩 | ●🟩 | ●🟩 | ● | ● | ● | ● |
| | Cronto | | [56] | | | ○🟥 | | ● | ○🟥 | ○ | 🟥 | 🟥 | ○🟥 | 🟥 | ● | ● | 🟥 | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ○🟩 | ●🟩 | ●🟩 | ● | ● | ● | ● |
| | MP-Auth | | [6] | | | ○🟥 | | ● | ○🟥 | 🟥 | ○🟥 | ○🟥 | ○🟥 | 🟥 | 🟥 | 🟥 | ● | | ○ | | | | | ●🟩 | ● | ● | ● | ● |
| | OTP over SMS | | | ●🟩 | ●🟩 | ○🟥 | | ● | 🟥 | ○ | ○🟥 | ○🟥 | 🟥 | 🟥 | ● | ● | ● | ●🟩 | ●🟩 | ●🟩 | ●🟩 | ○🟩 | ●🟩 | ●🟩 | ○🟥 | 🟥 | ● | ● |
| | Google 2-Step | | [57] | | | ○🟥 | | ● | ○🟥 | ○ | ○🟥 | ○🟥 | 🟥 | 🟥 | ● | ● | 🟥 | ○🟩 | ○ | ●🟩 | ●🟩 | | ●🟩 | ●🟩 | ● | ● | ● | ● |
| Biometric | Fingerprint | IV-I | [38] | ●🟩 | ●🟩 | ● | ○🟩 | ● | ○🟥 | 🟥 | 🟥 | ○🟥 | 🟥 | 🟥 | 🟥 | ○🟥 | 🟥 | ●🟩 | 🟥 | ●🟩 | | | | | 🟥 | ● | ● | 🟥 |
| | Iris | | [39] | ●🟩 | ●🟩 | ● | ○🟩 | ● | ○🟥 | 🟥 | 🟥 | ○🟥 | 🟥 | 🟥 | 🟥 | ○🟥 | 🟥 | ●🟩 | 🟥 | ●🟩 | | | | | 🟥 | ● | ○🟥 | 🟥 |
| | Voice | | [40] | ●🟩 | ●🟩 | ● | ○🟩 | ● | ○🟥 | 🟥 | 🟥 | ○🟥 | ○🟥 | 🟥 | ○🟥 | ○🟥 | 🟥 | ●🟩 | 🟥 | ○🟩 | | | | | 🟥 | ● | ● | 🟥 |
| Recovery | Personal knowledge | | [58] | ○🟩 | | ● | | ● | ● | ○ | ● | ● | ● | 🟥 | ● | ● | ● | | 🟥 | | | | | | ● | ● | ● | ● |
| | Preference-based | | [59] | ○🟩 | | ● | | ● | ○🟥 | ●🟩 | ○🟥 | ● | ● | 🟥 | ● | 🟥 | 🟥 | | ○ | | | | | ●🟩 | ● | ● | ● | ● |
| | Social re-auth. | | [60] | | | ● | | ● | 🟥 | ●🟩 | 🟥 | ● | ● | 🟥 | ● | ○🟥 | 🟥 | ○🟩 | 🟥 | ●🟩 | ●🟩 | ○🟩 | ○🟩 | ●🟩 | ● | 🟥 | ● | ○🟥 |

●= offers the benefit; ○= almost offers the benefit; *no circle* = does not offer the benefit.
🟩= better than passwords; 🟥= worse than passwords; *no background pattern* = no change.
We group related schemes into categories. For space reasons, in the present paper we describe at most one representative scheme per category; the companion technical report [1] discusses all schemes listed.

Table I
COMPARATIVE EVALUATION OF THE VARIOUS SCHEMES WE EXAMINED

## Brief comparison of different authentication methods

| | | Password | Biometric | CAP reader |
|---|---|---|---|---|
| usability | East-to learn | Yes | Yes | Yes |
| | Infrequent Errors | Quasi-yes | No | Quasi-yes |
| | Scalable-for-Users effort | No | Yes | No |
| | Easy recovery form loss of the authentication token | Yes | No | No |
| | Nothing to carry | Yes | Yes | no |
| deploy ability | Server-Compatible | Yes | No | No |
| | Brower-Compatible | Yes | No | Yes |
| | Accessible | yes | Quasi-yes | no |

## Brief comparison of different authentication methods

| | | Password | Biometric | CAP reader |
|---|---|---|---|---|
| Security | Resilient-to Physical-Observation | No | Yes | Yes |
| | Resilient-to-Targeted-Impersonation | quasi-yes | No | Yes |
| | Resilient-to-Throttled-Guessing | no | Yes | Yes |
| | Resilient-to-Unthrottled-Guessing | No | No | yes |
| | Resilient-to-Internal-Observation | No | No | Yes |
| | Resilient-to-Phishing | No | No | Yes |
| | No-Trusted-Third-Party | yes | Yes | Yes |
| | Resilient-to-Leaks-from-Other-Verifiers | no | no | yes |

- It seems that some set of goals are difficult to achieve at the same time, such as:
  - Memorywise-Effortless and Nothing-to-Carry
  - Memorywise-Effortless and Resilient-to-Theft

  Which is either the user remembers something, or it can be stolen (except for biometrics).
  - Server-Compatible and Resilient-to-Internal-Observation
  - Server-Compatible and Resilient-to-Leaks-from-Other-Verifiers.

  Server compatible means sending a password, where passwords can be stolen on user machine, replayed by one server to another.

# Multi-factor Authentication (MFA)

- Requires users to authenticate themselves using two or more authentication mechanisms.
- The mechanisms should involve different modalities.
  - Something you know
  - Something you possess
  - Something you are
- E.g. smart card and PIN, Username/password and token number.
- The idea is that the attacker must steal/subvert multiple authentication mechanisms to impersonate a user, i.e. the attacker might guess a password, but lack access to a user’s phone.

# Multi-factor Authentication (MFA)

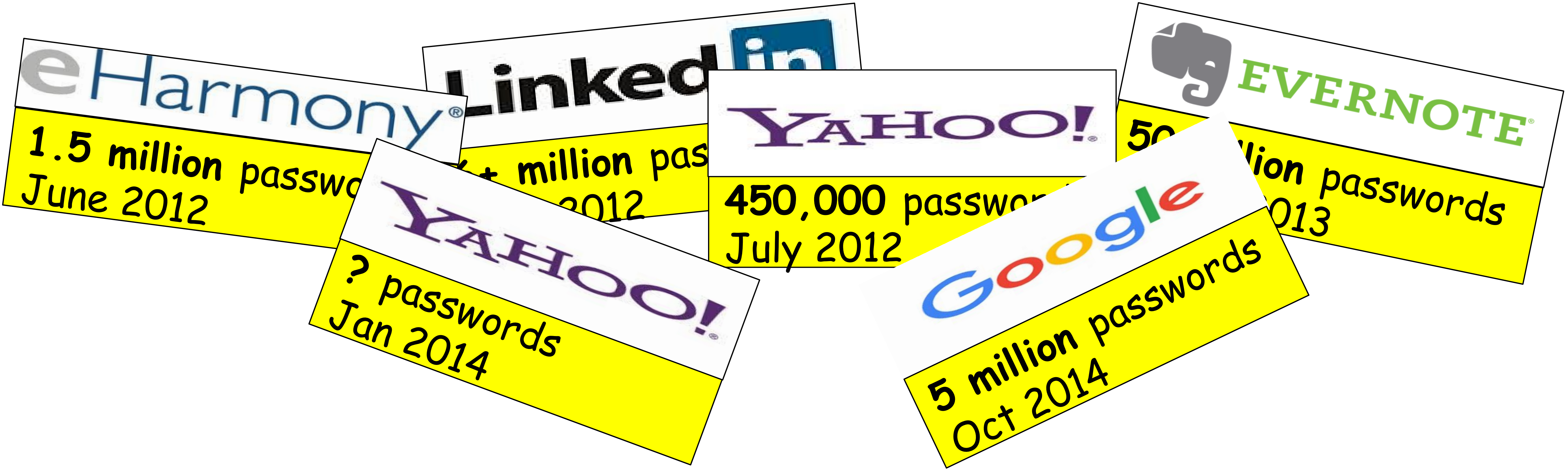
- Example: Google's two-factor authentication requires a password plus a cellphone which can receive authorization codes via text message.
- Example: Amazon Web Service (AWS) two-factor authentication requires a password and an MFA device.
  - MFA device can be asmartphone running an authentication app, or a special-purpose security token or security card.
- In general, MFA is good idea, however, empirical studies show that if users are given a second authentication factor in addition to passwords, users pick much weaker passwords.

## Honeywords: A cracking password detection

The slides and images of this section is from Honeywords: A New Tool for Protection from Password Database Breach by Kevin Bowers, Ronald L. Rivest, RSA conference 2014, power point
And the paper.

# Motivation-Good and bad news about password breaches

- The good news: when talking about password breaches, a convenient recent example is always available!



## Honeywords

- "Honeywords" proposed 2013 by Juels & Rivest

Honeywords: Making Password Cracking Detectable

- What is Honeywords? A method to make password-cracking detectable
  - Goal: detect password cracking, and improving the security of hashed password.
- How? Store additional false passwords (called honeywords) per each user.

# Honeywords

- Consider a system with n users $u_1,u_2, \ldots, u_n$, each have a password $p_1,p_2,\ldots,p_n$ respectively.
- System stores the user id $u_i$ with the hash of the password, pair $(u_i,H(p_i))$. Using salt is essential but will not effect the method, thus for simplicity we ignore it.
- Per each user, the system will add extra data (fake passwords) to the hash.
- If the adversary is able to have access to the password file, and is even able to extract all the password, he may not be able to exactly guess witch password is the real user password and which one is the fake.
  - If adversary tries one of them, a wrong password will detect the adversary!

## Attack scenarios

- Stolen files of password hashes: The adversary can steal the password file and apply offline brute-force attack or other methods and extract the passwords.
- Easily guessable passwords: The adversary tries to login by guessing password. Most of users choose weak passwords that enables a successful attack.
- Visible passwords: The adversary may get the password by observing the user entering the password.
- Same password for many systems or services: Mostly users use the same password for different services. So, if the password is broken on one system, the attacker is able to break other systems as well.
- Passwords stolen from users: Compromising end-devices such as laptops and phones using malwares help the adversary to steal user password.
- Password change compromised: The mechanism in which the user is able to change/recover its password is compromised, thus the attacker will gain access to user password.

## Honeychecker

- The paper focus on the first scenario, in which the attacker has access to hashed password file.
- There is the Honeychecker server which is a separate hardened computer system where secret information can be stored.
  - Create a distributed security, since the computer which stores the password file is different from the Honeychecker.
  - Compromising of Honeychecker, in worst case will only reduce the security to the level before using Honeywords method.
- Assumption: computer system can communicate with the honeychecker when a login attempt is made on the computer system, or when a user changes her password. In addition, the communication is over dedicated lines and/or encrypted and authenticated.
- Depending on the policy chosen, honeychecker may
  - signal to the computer system that login should be denied
  - merely signal a “silent alarm” to an administrator, and let the login on the computer system proceed

## Honeychecker

- Has a single database value c(i) for each user $u_i$. The values are integers in the range 1 to k, for some small integer parameter k (e.g. k = 20).
- Two type commands:
  - Set(i, j): Sets c(i) to have value j.
  - Check(i, j): Checks that c(i) = j. Return result of check to requesting computer system or raise an alarm if check fails (base on configuration).

## Approach – Setup

- For each user $u_i$, a list $W_i$ of distinct words, called “sweetwords”
$$W_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,k})$$
- Only one of these sweetwords $w_{i,j}$ is equal to the password $p_i$ known to user $u_i$.
- c(i) denote the index of user $u_i$’s password in the list $W_i$, so that
$$w_{i,c(i)} = p_i$$
  - sugarword : The correct password $w_{i,c(i)}$ .
  - Honeywords: Other (k − 1) words $w_{i,j}$.
- Toughnut: a very strong password whose hash the adversary is unable to invert.
- Some honeywords may be “*toughnut*”.

# Approach – Setup

- Current password file F will have new format. Instead of containing pairs $\left(u_i, H(p_i)\right)$, in contains pairs $(u_i, H_i)$, where

$$v_{i,j} = H\left(w_{i,j}\right)$$
$$H_i = (v_{i,1}, v_{i,2}, \dots, v_{i,k})$$

  - Note: The user only needs (as usual) to remember her password $p_i$.

- $\text{Gen}(k, p_i)$: the procedure used to generate both a list $W_i$ of length k of sweetwords for user $u_i$ and an index c(i) of the correct password $p_i$ within $W_i$:

$$\text{Gen}(k, p_i) = (W_i, c(i))$$

## Approach – Login

- Login system should determine whether a proffered password *g* is equal to the user's password or not.
  - If *g* is not the user's password, the login routine needs to determine whether *g* is a honeyword or not.
    - If the user/adversary submitted the correct password for the user, then login proceeds successfully as usual.
    - If the adversary has entered one of the ***user's honeywords***, obtained for example by brute-forcing the password file F, then an appropriate action takes place (determined by policy), such as
      - setting off an alarm or notifying a system administrator,
      - letting the login proceed, but on a honeypot system,
      - tracing the source of the login carefully, etc.

## Approach – Login

- How does the login routine determine whether $g$ = $p_i$?
  - If the hash H(g) of $g$ is not in the file F for the user $u_i$, then word $g$ is neither the user's password nor one of the user's honeywords, so login is denied.
  - If hash H(g) of $g$ matches one of the sweetword in $W_i$ list for user $u_i$, the login routine should detect whether $g$ is the user's password, or one of the user's honeywords.
- Base on the matching $\mathrm{H(g)} = v_{i,j}$, the login routine will know the index of sweetword that matches g. But it doesn't know whether
$$j = c(i)$$
- The computer system sends the honeychecker the following message
$$check(i, j)$$
- The honeychecker determines whether $j = c(i)$, if not, an alarm is raised and other actions may be taken.

## Approach – Change of password

- When user $u_i$ changes password, or her account is first initialized, the system needs to:
  - use procedure $\mathrm{Gen}(k, p_i)$ to obtain a new list $W_i$ of k sweetwords, the list $H_i$ of their hashes, and the value c(i) of the index of the correct password $p_i$ in $W_i$.
  - securely notify the honeychecker of the new value of c(i), and
  - update the user's entry in the file F to $(u_i, H_i)$.
- Note: The honeychecker does not learn the new password or any of the new honeywords.
- Finally, the computer system sends the honeychecker following message:

$$Set(i, j)$$

## Security Definitions

- Consider z be the adversary's expected probability of winning the game. This probability is taken over the user's choice of password $p_i$, the generation procedure Gen(k; $p_i$), and any randomization used by the adversary to produce its guess j. Thus z ≥ 1/k, since an adversary can win with probability 1/k just by guessing j at random.
- *Flatness:* Honeyword generation method is $\epsilon - flat$ for a parameter $\epsilon$ if the maximum value over all adversaries of the adversary's winning probability z is $\epsilon$.
  - *perfectly flat*: If the generation procedure is as flat as possible (i.e., 1/k flat).
  - approximately flat: If it is $\epsilon - flat$ for $\epsilon$ not much greater than 1/k.
- For k=20; if *Gen* is perfectly flat, the adversary who has compromised F and inverted H has a chance of at most 5% of picking the correct password $p_i$ from this list. In this ideal case, $\epsilon = 1/20$.

## Honeyword Generation

- Different flat (or approximately flat) generation procedures Gen
  - With legacy-UI (User Interface) procedures, the password-change UI is unchanged.
    - Chaffing-by-tweaking
      - chaffing-by-tail-tweaking
      - chaffing-by-tweaking-digits
    - chaffing-with-a-password-model
  - With modified-UI procedures, the password-change UI is modified to allow for better password/honeyword generation.
    - take-a-tail

## Legacy-UI password changes

- In this method, the password-change procedure asks the user for the new password.
- The UI does not tell the user about the use of honeywords, nor interact with her to influence her password choice.
- Note: It is possible to change the honeyword generation procedure without needing to notify the user or to change the UI.
- In this method the (k-1) honeywords are generates ***similar in style*** to the password $p_i$ given by the user $u_i$, so that an adversary will have difficulty in identifying $p_i$ in the list $W_i$ of all sweetwords for user $u_i$.
- *Chaffing*: The honeyword generation procedure Gen(k, $p_i$) or "cha ff procedure" generates a set of (k-1) distinct honeywords ("chaff").
  - Note that the honeywords may depend upon the password $p_i$.
  - The password and the honeywords are placed into a list $W_i$, in random order. The value c(i) is set equal to the index of $p_i$ in this list.

## Legacy-UI password changes

- The success of chaffing depends on the quality of the chaff generator.
- Two basic methods for chaffing:
  - *Chaffing by tweaking*
  - *Chaffing-with-a-password-model*

# Chaffing by tweaking

- In this method some character positions of the password are selected in order to obtain the honeywords.
- Let t denote the desired number of positions to tweak.
- The honeywords are then obtained by tweaking the characters in the selected t positions: each character in a selected position is replaced by a randomly-chosen character of the same type: digits are replaced by digits, letters by letters, and special characters (anything other than a letter of a digit) by special characters.
- *chaffing-by-tail-tweaking*: the last t positions of the password are chosen to be changed in order to obtain the honeywords.
- *chaffing-by-tweaking-digits*: only the last t digits will be changed for honeywords production.

## Chaffing by tweaking

- Example 1
  - User password: BG+7y45 (called sugar)
  - T=3, k=4
  - Method: *chaffing-by-tail-tweaking*
  - ***Possible honeywords***: BG+7q03, BG+7m55, BG+7y45, BG+7o92 (called honey).
- Example 2
  - User password: 40*flavors
  - T=2, k=2
  - Method: *chaffing-by-tweaking-digits*
  - ***Possible honeywords***: 42*flavors, 57*flavors, 18*flavors.

## Chaffing-with-a-password-model

- uses a probabilistic model of real passwords; this model might be based on a given list L of thousands or millions of passwords and perhaps some other parameters. This method does not necessarily need the password in order to generate the honeywords, and it can generate honeywords of widely varying strength.
- Example of list of 19 honeywords generated by one simple mode:

kebrton1

a71ger

#NDYRODD_!!

venlorhan

pizzhemix01

9,50PEe]KV.0?RIOtc&L-:IJ"b+WoI<*[!NWT/pb

mobopy

02123dia

forlinux

sbgo864959

aj1aob12

sveniresly

WORFmgthness

# Chaffing-with-a-password-model

- *Modeling syntax:* Bojinov et al. propose the following approach to chaffing-with-a-password model. In their scheme,
  - password is parsed into a sequence of “tokens”
  - each token represents a distinct syntactic element (either a word, number, or set of special characters).
  - Honeywords are then generated by replacing tokens with randomly selected values that match the tokens.
- Example
  - User password: mice3blind
  - token sequence$W_4$ |$D_1$ |$W_5$ (where D represent digit, W word and the index numbers shows the length)
- Generation approach creates: $W_4 \leftarrow$ gold, $D_1 \leftarrow 5$ , $W_5 \leftarrow$ rings, thus the honeyword is ***gold5rings***

## Modified-UI password changes

- take-a-tail: This method is identical to the chaffing-by-tail-tweaking method, except that the tail of the new password is now randomly chosen by the system, and required in the user-entered new password.
- That is, the password-change UI is changed from:

  Enter a new password:

  - to something like:

  Propose a password: • • • • • • •

  Append '413' to make your new password.

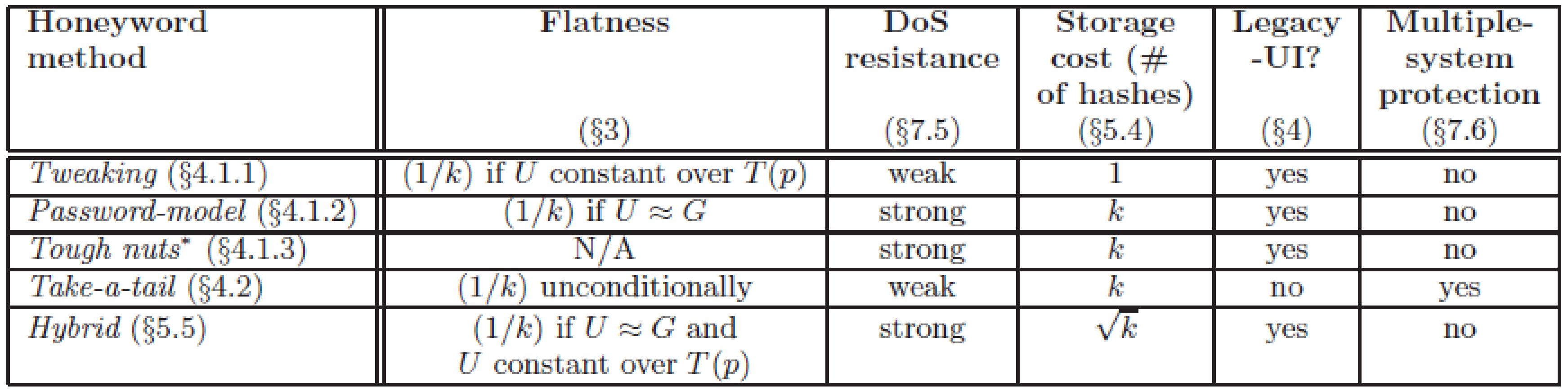
  Enter your new password: • • • • • • • • • •

- Thus, if the user proposes "RedEye2," his new password is "RedEye2413."

Which means the user is forced by the system to add extra data to its password and memorize it as part of her password.

- *Once the password has been determined, the system can generate honeywords in same manner as **chaffing-by-tail-tweaking**.*

# Comparison of methods



| Honeyword method | Flatness (§3) | DoS resistance (§7.5) | Storage cost (# of hashes) (§5.4) | Legacy -UI? (§4) | Multiple-system protection (§7.6) |
|---|---|---|---|---|---|
| *Tweaking* (§4.1.1) | $(1/k)$ if $U$ constant over $T(p)$ | weak | 1 | yes | no |
| *Password-model* (§4.1.2) | $(1/k)$ if $U \approx G$ | strong | $k$ | yes | no |
| *Tough nuts** (§4.1.3) | N/A | strong | $k$ | yes | no |
| *Take-a-tail* (§4.2) | $(1/k)$ unconditionally | weak | $k$ | no | yes |
| *Hybrid* (§5.5) | $(1/k)$ if $U \approx G$ and $U$ constant over $T(p)$ | strong | $\sqrt{k}$ | yes | no |

## Variations and Extensions

- "Random pick" honeyword generation
- Typo-safety
- Storage optimization
- Hybrid generation methods

## “Random pick” honeyword generation

- It is a modified-UI procedure that is perfectly flat, where first all the k element of the Wi list is generated in some arbitrary manner (which may involve interaction with the user) and then one of them is picked uniformly at random as the new password; the other elements become honeywords.  As an example of user involvement, the system might ask the user for k potential passwords. The value c(i) is set equal to the index of (randomly chosen) password $p_i$ in this list.
- This is completely flat no matter how the list is generated.
- It is probably a bad idea to ask the user for k sweetwords, since the user may remember and mistakenly enter a sweetword supplied by her and used by the system as a honeyword.

## Typo-safety

- In order to prevent the rare accidents, such as legitimate user set off an alarm by accidentally entering a honeyword, some typo-safety may be considered.
- In tail-tweaking it would be helpful if the password tail were quite different from the honeyword tails, so a typing error won't turn the password into a honeyword.
- Use an error-detection code to detect typos. Let q denote a small prime, then any two sweetword tails must be multiple of q.

## Storage optimization

- We can optimize some honeyword generation methods, such as tweaking and take-a-tail, to reduce their storage to little more than a single password hash.
- Consider tailt-weaking where the tails are t-digit numbers and let T (p) denote the class of sweetwords obtainable by tweaking p for the selected character positions.
  - 1) k=|T(pi)|
  - 2) let $W_i$=$T(p_i)$=($w_{i,1}, w_{i,2}, \dots, w_{i,k}$), sorted into increasing order lexicographically.
  - 3) select a random element $r \in \{0,1,\dots,k-1\}$ uniformly at random.
  - 4) compute and store $H(w_{i,r})$.
- To verify a proffered password g,
  - 1) The computer system computes the hash of each sweetword in T (g)
  - 2) If one is found equal to $H(w_{i,r})$ then $w_{i,r}$ is known, which means the set$W_i$ is correct
  - 3) Find the position j of g in $W_i$.
  - 4) The honeychecker operates as usual: The computer system sends j to the honeychecker to check whether j = c(i).

## Hybrid generation methods

- It is possible to combine the benefits of different honeyword generation strategies by composing them into a “hybrid” scheme.
  - Use chaffing-with-a-password-model on user-supplied password p to generate a set of a ($\geq$2) seed sweetwords W’, one of which is the password. Some seeds may be “tough nuts.”
  - Apply chaffing-by-tweaking-digits to each seed sweetword in W’ to generate b ($\geq$ 2) tweaks (including the seed sweetword itself). This yields a full set W of $k = a \times b$ sweetwords.
  - Randomly permute W. Let c(i) be the index of p such that p = wc(i), as usual.

## Policy Choices

- Password Eligibility: Some words may be ineligible as passwords because they violate one or more policies regarding eligibility, such as:
  - Password syntax: A password may be required to have a minimum length, a minimum number of letters, a minimum number of digits, and a minimum number of special characters. The initial character may be restricted.
  - Dictionary words: A password may not be a word in the dictionary, or a simple variant thereof.
  - Password re-use: A password may be required to be different than any of the last r passwords of the same user, for some policy parameter r (e.g. r=10).
  - Most common passwords: A password may not be chosen if it appears on a list of the 500 most common passwords in widespread use (to prevent online guessing attacks).
  - Popular passwords: A password may not be chosen if m or more other users in a large population of users are currently using this password.

## Policy Choices

- Failover: The computer system can be designed to have a “failover” mode so that logins can proceed more-or-less as usual even if the honeychecker has failed or become unreachable.
  - In failover mode, honeywords are temporarily promoted to become acceptable passwords; this prevents denial-of-service attacks resulting from attack on the honeychecker or the communications between the system and the honeychecker.
  - The cost in terms of increased password guessability is small. Temporary communication failures can be addressed by buffering messages on the computer system for later delivery to and processing by the honeychecker.

## Policy Choices

- Per-user policies: policies may vary per-user (this is not uncommon already).
  - Honeypot accounts: Such accounts can help identify theft of F and distinguish over a DoS attack. Which accounts are honeypot accounts would be known only to the honeychecker.
  - Selective alarms: Raise an alarm if there are honeyword hits against administrator accounts or other particularly sensitive accounts, even at the risk of extra sensitivity to DoS attacks. Policies needn't be uniform across a user population.
- Per-sweetword policies: In this policy, the Set(i, j) command to the honeychecker has an optional third argument ai,j, which says what action to take if a Check(i, j) command is later issued.
  - The actions might be of the form " Raise silent alarm", "Allow login" or "Allow for single login only" , etc.

## Attacks

- The paper studies various attacks possible against the methods proposed:
  - General password guessing:
    - Legacy-UI methods don't affect how users choose passwords, so they have no beneficial effect against adversaries who try common passwords in an online guessing attack.
    - Modified-UI methods like take-a-tail also affect the choice of password—appending a three-digit random tail to a user chosen password effectively reduces the probability of the password by a factor of 1000.
  - Targeted password guessing: Personal information about a user could help an adversary distinguish the user's password from her honeywords. It is often feasible to deanonymize users, that is, ascertain their real-world identities, based on their social network graphs or just their usernames. Given a user's identity, there are then many ways to find demographic or biographical data about her online—by exploiting information published on social networks, for example. Knowing a user's basic demographic information, specifically his/her gender, age, or nationality, is known to enable slightly more effective cracking of the user's hashed password. Similarly, attackers often successfully exploit biographical knowledge to guess answers to personal questions in password recovery systems and compromise victims' accounts
    - As chaffing-with-apassword-model creates honeywords independently of user's password, this method of honeyword generation may enable adversaries to target data-mining attacks against users and gain some advantage in distinguishing their passwords from their honeywords.

## Attacks

- Attacking the Honeychecker: The adversary may decide to attack the honeychecker or its communications with the computer system.
  - The updates (“Set” commands) sent to the honeychecker need to be authenticated, so that the honeychecker doesn’t incorrectly update its database.
  - The requests (“Check” commands) sent to the honeychecker also need to be authenticated, so that the adversary cannot query the honeychecker so as to cause an alarm to be raised.
  - The replies from the honeychecker should be authenticated, so that the computer system doesn’t improperly allow the adversary to login.
  - By disabling communications between the computer system and the honeychecker, the adversary can cause a failover. The computer system then either has to disallow login or take the risk of temporarily allowing login based on a honeyword and buffering messages for later processing by the honeychecker.
  - The deployment of the computer system and the honeychecker as two distinct systems itself brings the usual benefits of separation of duties in enhancing security. The two systems may be placed in different administrative domains, run different operating systems, and so forth.

# Attacks

- Likelihood Attack: If the adversary has stolen F and wishes to maximize his chance of picking $p_i$ from $W_i$, he can proceed with a "likelihood attack" as follows.
  - Assume here that we are dealing with an approach based on generating honeywords using a probabilistic model.
  - Let G(x) denote the probability that the honeyword generator generates the honeyword x.
  - Let U(x) denote the probability that the user picks x to be her password.
  - Let $W_i = \{w_{i,1}, \ldots, w_{i,k}\}$.
  - The likelihood that c(i) = j, given $W_i$, is equal to

$$U(w_{i,j}) \prod_{j' \neq j} G(w_{i,j'}) = CR(w_{i,j})$$

  - Where

$$C = \prod_{j'} G(w_{i,j'})$$

  - And where

$$R(x) = U(x)/G(x)$$

the relative likelihood that the user picks x compared to the honeyword generator picking x.

## Attacks

- Denial-of-service:
  - Is a potential problem for methods such as chaffing-by-tweaking that generate honeywords by predictably modifying user supplied passwords.
  - The concern is that an adversary who has not compromised the password file F, but who nonetheless knows a user's password, can feasibly submit one of the user's honeywords.
  - An overly sensitive system can turn such honeyword hits into a DoS vulnerability. One (drastic) example is a policy that forces a global password reset in response to a single honeyword hit.
  - Conversely, in a system inadequately sensitive to DoS attacks, an adversary that has stolen F can guess passwords while simulating a DoS attack to avoid triggering a strong response. A policy of appropriately calibrated response is important and can reduce DoS attacks' potency.
- Multiple systems: As users commonly employ the same password across different systems, an adversary might seek an advantage in password guessing by attacking two distinct systems, system A and system B—or multiple systems, for that matter.

## Conclusion

- User authentication factors.
- Passwords are the most popular.
- Verity of attacks on passwords
- Passwords are preferred to be used since they have strong deployability and usability.
- Honeywords, a detection password-cracking method

## Thank you

# How to calculate password entropy?

- Consider password as strings of English words, where words are randomly selected from a collection of 2048 words.
  - The entropy for each word is $\log(2^{11}) = 11.$
  - Password entropy= $11 \times number\ of\ words\ in\ the\ password$
- NIST entropy: Entropy for random password with length l that is selected from a keyboard set of 94 printable characters is about $= log_2(94^l) = 6.55 \times l.$

back

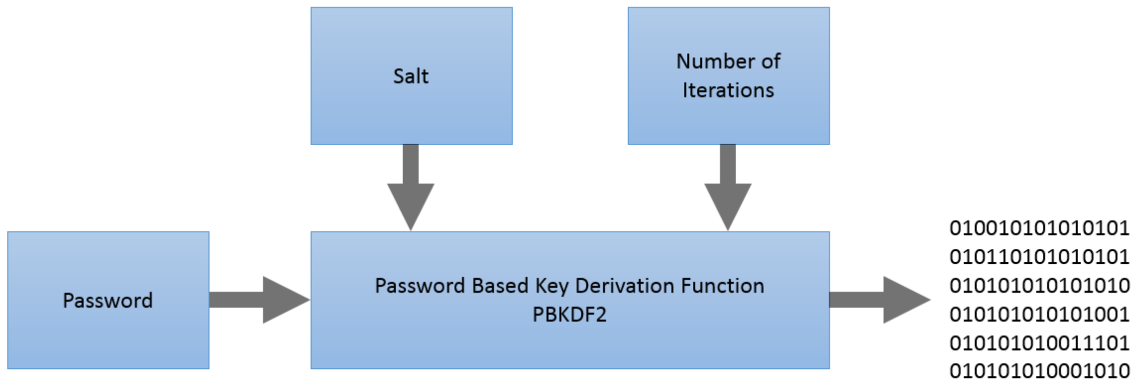
## Hash function optimization

- The computation of hash algorithm can be optimized (less work to be done) by the adversary in order to guess the password faster.
- Why? Hash algorithms are not designed for password protection. However they are used to protect passwords.
- General techniques:
  - Zero-based optimizations
  - Early-exit optimizations
  - Initial-step optimizations

back

## Key-derivation function

- In cryptography, a key derivation function (KDF) derives one or more secret keys from a secret value such as a master key, a password, or a passphrase using a pseudo-random function.
  - A Password-Based Key Derivation Function (PBKDF2) is a key derivation function that is part of the RSA Public Key Cryptographic Standards series.
    - A PBKDF2 takes a password, a salt to add additional entropy to the password, and a number of iterations value. The number of iterations value repeats the hash operation over the password multiple times to produce a derived key for the password that can be stored in a database.
    - By repeating a hash or encryption process over the password multiple times, you are algorithmically slowing down the hashing process which makes brute force attacks against the password much harder. This means that fewer passwords can be tested at once.



Source : https://www.microsoft.com

back

# Rainbow table

- random set of *initial passwords* from P (finite set of passwords)
- Applies alternating the hash function with the reduction function(create chain).

$$\text{aaaaaa} \xrightarrow[H]{} \text{281DAF40} \xrightarrow[R]{} \text{sgfnyd} \xrightarrow[H]{} \text{920ECF10} \xrightarrow[R]{} \textbf{kiebgt}$$

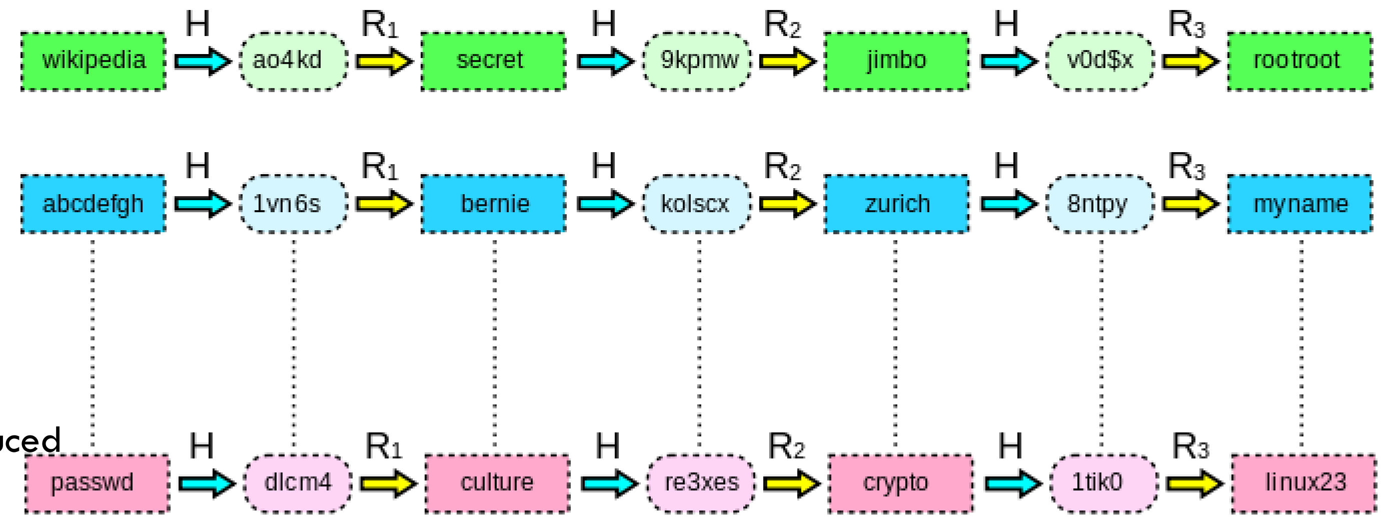
- Example: hash 920ECF10, first applying R

$$\text{920ECF10} \xrightarrow[R]{} \textbf{kiebgt}$$

- the password is sgfnyd
- *false alarm*

$$\text{FB107E70} \xrightarrow[R]{} \text{bvtdll} \xrightarrow[H]{} \text{0EE80890} \xrightarrow[R]{} \textbf{kiebgt}$$

- Extend the chain to find another match
- If is extended to length k with no match, result: never produced by the adversaries rainbow table



Source: https://en.wikipedia.org/wiki/Rainbow_table

back

## Secure Remote Password protocol

- The Secure Remote Password protocol is a method in which an eavesdropper or man-in-the-middle cannot obtain enough information to be able to brute force guess a password without further interactions with the parties for each guess.
- Notation

$n$ : A large prime number. All computations are performed modulo $n$.

$g$ : primitive root modulo $n$ (often called a *generator*).

$s$ : A random string used as the user's salt.

$P$ : The user's password.

$x$ : A private key derived from the password and *salt*.

$v$ : The host's password verifier.

$u$ : Random scrambling parameter, publicly revealed.

$a, b$ : Ephermeral private keys, generated randomly and not publicly revealed.

$A, B$ : Corresponding public keys.

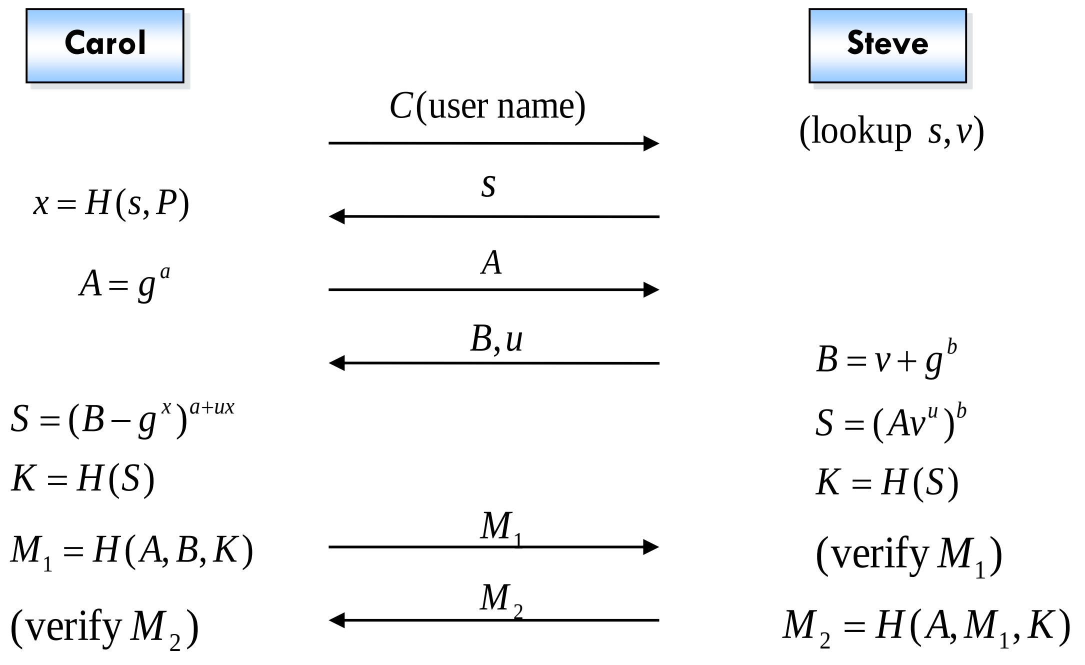
$H()$ : One - way hash function.

$K$ : Session key.

back

## Secure Remote Password protocol

- Protocol
  - To establish a password P with Steve, Carol picks a random salt s, and computes $x = H(s,P),\ v = g^{x}$.



Source: power point "The Secure Password-Based Authentication Protocol" by Jeong Yunkyoung