
Passwords and Authentication

Steven M. Bellovin

`https://www.cs.columbia.edu/~smb`



Authentication Principles

- There's a conventional trilogy: something you know, something you have, something you are
- That's correct, as far as it goes—but it ignores the *systems* nature of authentication
- Systems nature: multiple pieces interact; security requires that all components be secure, and that their composition be secure.
- You cannot fix authentication by changing only one piece

Passwords: Something You Know

- A classic means of authentication
- Everyone understands them
- They're cheap to deploy
- However...

The Trouble with Passwords

- People forget them
- People pick weak passwords
 - ☞ Common passwords, in one recent analysis, included “password”, “12345”, and “123456”
- People share them
- People write them down
- Attackers can replay them

“Pick Strong Passwords”

- In 1979, Morris and Thompson warned us about weak passwords
- They were working with hardcopy terminals with no computational ability
- No keystroke loggers or phishers
- Most users had one or two passwords; almost no one had more than a small handful
- By actual count, I have more than 200 web passwords. . .



(Photo courtesy Perry Metzger)

Old Threats versus New Threats

- Hacker steals hashed system password file from timesharing machine
- Attacker has limited CPU resources for cracking it

Old Threats **versus New Threats**

- Hacker steals hashed system password file from timesharing machine
- Attacker has limited CPU resources for cracking it
- Hacker steals application—not system—password file from web server
- May be plaintext, for password recovery
- Secondary authentication questions are jokes—too easy for a targetier to figure out
- Malware plants keystroke loggers
- Users are lured to phishing websites
- Attacker has botnets, GPUs, cloud services

The Advice Hasn't Changed

- “Pick strong passwords”—at least three letters, three digits, one symbol, and one hieroglyphic or character from Klingon
- “Never write them down”
- “At least 8 characters”

Even the Strength Rules Aren't Great

Rules from a US Government website

- Minimum Length: 8
- Maximum Length: 12
- Maximum Repeated Characters: 2
- Minimum Alphabetic Characters Required: 1
- Minimum Numeric Characters Required: 1
- Starts with a Numeric Character
- No User Name
- No past passwords
- At least one character must be from

~ ! @ # \$ % ^ & * () - _ + ! + = { } [] \ | ; : / ? . , < > " ' ` !

Attacking this Scheme

- The first character must be numeric, so there are only ten choices.
- Only one digit is required; most people will use a consecutive string of digits followed by a consecutive string of letters.
- A punctuation character is needed, but most people will just put a period at the end.
- Likely pattern: one or more digits, one or more letters, and a period, where the total number of digits and letters will be seven.
- Total combinations somewhat less than $\sum_{i=1}^6 10^i \cdot 26^{7-i}$ which comes to 5,003,631,360.
- Easily attackable

What Has Changed?

- Morris and Thompson threat model: someone (possibly an insider) grabs the system password file, but has limited resources to use for a guessing attack
- Today: phishing, keystroke loggers, subverted systems—none of which are bothered by strong passwords
- Attackers have vast computational resources
- Most logins are for web sites—people have many web site logins, and web sites have far more users than the largest systems from 1979
- Why should we expect the same defenses to work?

Phishing

- Phishing sites don't care about strength rules
- The very first phishing message I saw was from **paypa1 . com**

Let Me Enlarge That and Change the Font

paypa1.com

versus

paypal.com

The threat model has changed!

Cyrillic Homograph Attack on “Paypal”

Glyph	Unicode value in Cyrillic
Р	U+0420
а	U+0430
у	U+0443
р	U+0440
а	U+0430
l	U+006C (ASCII)

Some symbols look the same, but have different values: ordinary /—technically called “solidus”—is U+002F, but U+2044, “fraction slash”, looks the same. Think about that in a URL. . .

Employees and Users

- You can train employees, and *insist* on certain behaviors
- (Of course, employees can be stubborn, and “change” their passwords by incrementing a digit, switching among two or three favorite passwords, etc.)
- If user—that is, *customer*—password requirements for your web site are too annoying, they’ll shop somewhere else
- (Password strength requirements correlate more with lack of choice (employer, government) than with the assets at risk)

Mandating Password Changes

- People are stubborn, and rarely cooperate properly
- People often forget new passwords, which forces more reliance on secondary authentication—and that's generally very weak
- Experiments have shown that new passwords can be derived algorithmically from old ones, with reasonably high accuracy—41% overall, and 17% in 5 or fewer guesses
- The rationale for frequent changes is poor—or rather, it's set out in an equation, but the values to plug in are unknowable, and don't take today's threat model into account

What is the Threat Model?

- Online or offline guessing attacks?
- Are the attacks targeted or random?
- What are the enemy's abilities?

The classic “just pick strong passwords” defense gets most of this wrong!

The Password Dilemma

- We each have very many logins
- We need to use strong passwords for them
- Reusing passwords is *very* dangerous
- We can't possibly remember them all
- They have to be stored—somehow

Storing Passwords: Users

- Requirements: security, stability, usability—these can conflict. . .
- A piece of paper is usually usable and usually secure—unless you’re being targeted by a high-level adversary
- ☞ However—not very usable or very secure for mobile devices, and often hard to back up
- Higher-tech version: *password manager*
- For usability, the password store should be “near” the browser; for security, it should be away from it

Password Managers

- Specialized programs that store per-site passwords
- Storage is generally encrypted
- Many managers will synchronize passwords between different devices, including mobile devices
 - Cloud storage?
 - USB device?
 - Special LAN protocol?
 - Which are secure? Which are convenient?
- Many are integrated with browsers

Password Managers and Browsers

- Many password managers use browser extensions
- Very convenient—they can autofill passwords on login pages
- Protects against most phishing attacks—they'll only supply the password for the correct site
- ☞ These are the advantages of “near the browser”
- But—if the browser is compromised, the attack code can probably get at the passwords via the manager's browser extension
- ☞ *This is the problem with “near”*

Lost Passwords

- If you run a production site, some users *will* forget their passwords
- You have to provide some way to recover—but how you do it is heavily dependent on the threat model and the operational model
- If people (e.g., your help desk or your customer support line) are involved, be extremely careful about the procedures they follow, and do not let them deviate.

Secondary Authentication

- Questions must be memorable to the users, but hard to find by attackers
- Very challenging, in an age of social networks, and harder for public figures
- The classic question—“What is your mother’s maiden name?”—was used at least as early as 1882:

**Identity can be established if the party will
answer that his or her mother’s maiden name
is.....) 05626 Guineapig**

(from an old telegraph codebook)

- Targetiers can learn the answer fairly easily, but in many places, marriage and birth records are online; it’s not hard to automate the guessing process

Reset or Recovery?

- Always reset the password; don't send the original
- You can only send the original if you have it, and that's very dangerous
- Often, the new password is emailed to the user—ok for modest-value accounts, but it makes the email password the most valuable one the user has
- Reset important passwords by paper mail, text message, or even in-person
- Rarely much reason to force the user to change the password after reset—unless it's a higher-value account and you don't want to risk the new one being stored in the user's mailbox

Password File Compromise

- The odds on it happening are moderately high—it's happened to enough sophisticated companies that there's little reason to think you're safe
- You must reset all passwords
- Were your secondary authentication answers compromised?
- How were the passwords stored?

Storing Passwords: Servers

- *Never* store plaintext passwords—if you do, any compromise is a disaster
- One site apparently used encryption (in ECB mode!); this is almost as bad, since the attacker can probably steal the decryption key, too
- First decent approach: hash the password
- Server takes the user-supplied password, hashes it, and compares against the stored value.
- Better, yet, “salt”, hash, and iterate

Basic Attack on Hashed Passwords

```
for i in large_dictionary:
    for j in variants_of(i):
        if H(j) in stolen_password_file:
            print j, stolen_password_file[j]
```

Why Iterate?

- The attacker has some guess rate n tries/second
- If you use $H^m(\text{password})$ instead of $H(\text{password})$, you cut the guess rate to n/m
- The attacker can make fewer guesses per unit time
- Guesses are cheap, but $n \neq \infty$


What is Salt?

- Pick a random number s
- Calculate $H^m(\text{password}||s)$; store that and s
- It stops attackers from precalculating a hashed dictionary
- It hides that fact that two hashed passwords correspond to the same plaintext
- (The Morris/Thompson design used a 12-bit salt, which was fine when a large site had a few hundred users. A large site today has 10s of millions—and Facebook has over a billion. I'd recommend at least 64 bits of salt today.)

Which Hash Function? How Many Iterations?

- The goal is to slow down guessing attacks
- Any non-invertible function will suffice; if it's too fast, iterate longer
- (Yes, MD5 is fine.)
- Caveat: don't use a function that limits input length or character set

Lamport's Algorithm

- Conventional passwords are replayable—this is at the heart of phishing attacks. We can do better
 - Server stores $n, H^n(\text{password})$. User enters $H^{n-1}(\text{password})$
 - Server calculates $H(H^{n-1}(\text{password}))$; if it matches, the entered value and $n - 1$ are stored for next time
 - The user could have local computational capability—or the user can print out $H^{n-1}, H^{n-2}, \dots, H^{n-i}$ and have i passwords to use on a trip
 - These passwords are not replayable (but guessing attacks are still possible)
 - Note: inherent limit to the number of logins possible before a password change, determined by the initial value of n
-  The math requires periodic password changes. . .

One-Time Passwords

- Lamport's algorithm is a form of *one-time password*: a password that's usable only once
- Most other forms require some sort of device or token: “something you have”
- Cryptography is sometimes used, but not always
- Generally used with a PIN or password, to guard against misuse of stolen devices.

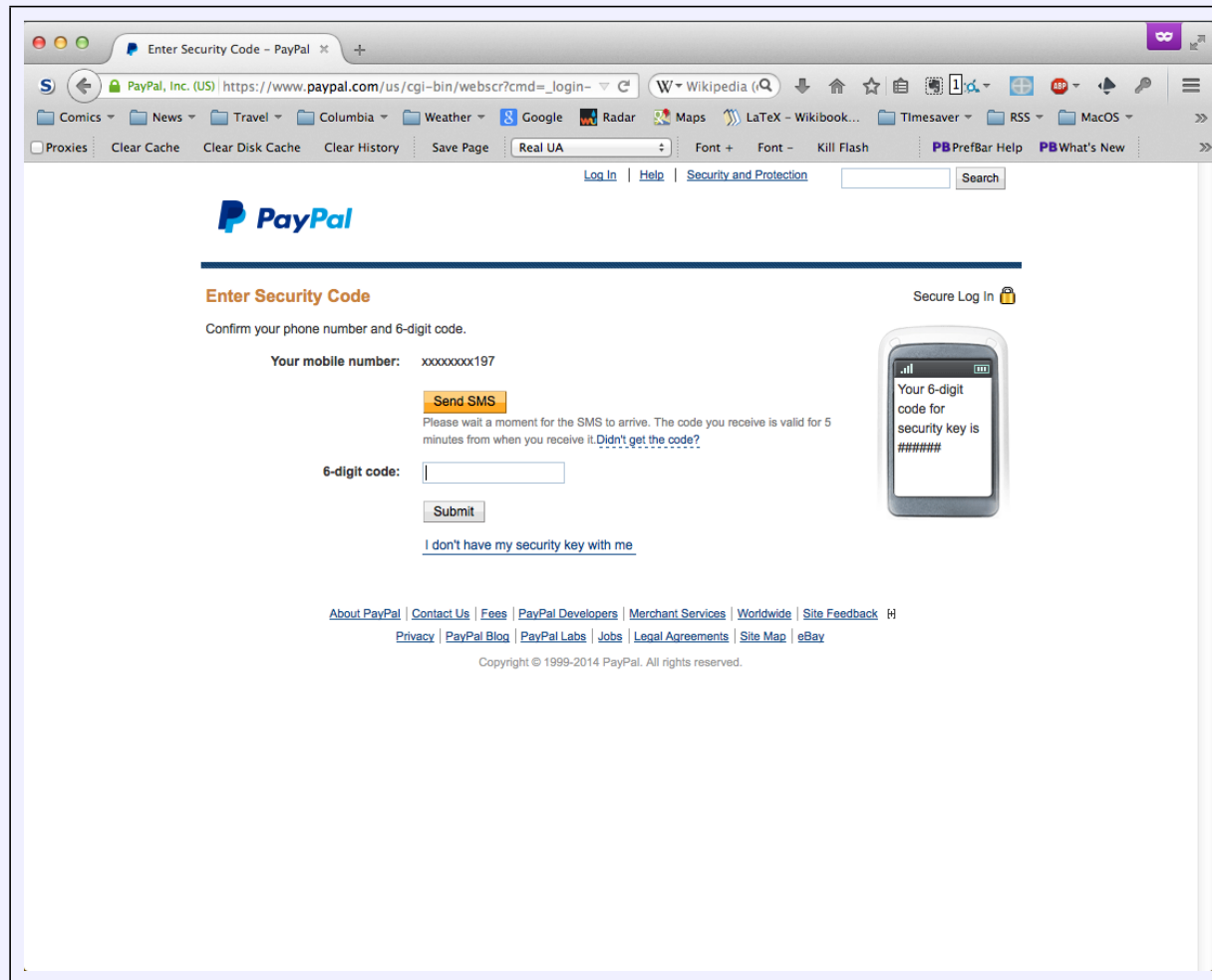
Advantages of Tokens

- Guessing attacks don't work—they use strong secrets
- If one is lent out, the proper owner doesn't have it
- Generally speaking, the authentication code isn't repayable
- But—can be lost, reverse-engineered, etc.

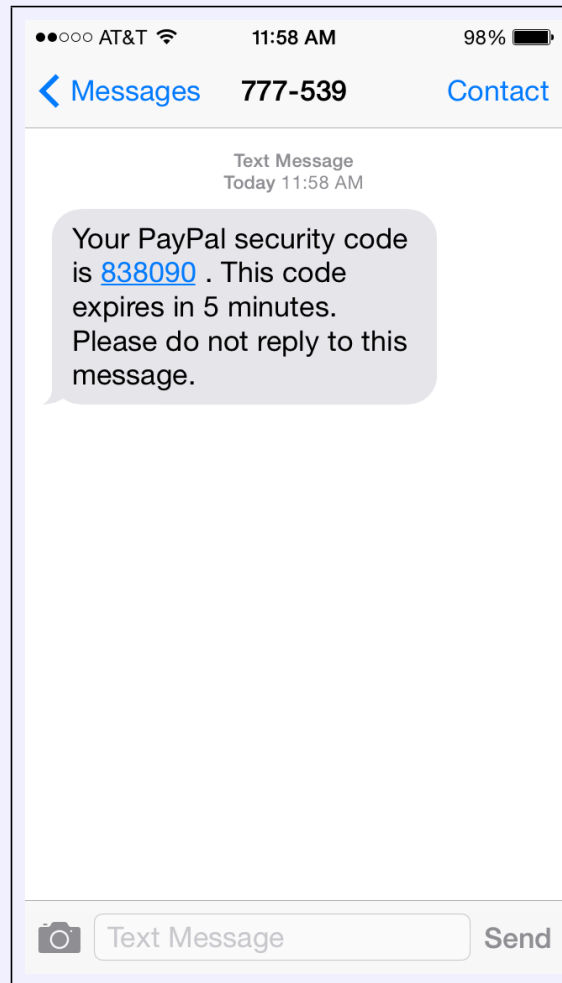
Challenge-Response Authentication

- The server sends x to the user; the user replies with $F(x)$
- Common cryptographic form of challenge-response: $F(x) = E_k(x)$, where E_k is encryption with a key k known to the server and to the user's device
- Non-cryptographic variant: send the challenge as a text message to the user's mobile phone

Paypal Login Screen



Paypal Challenge



Phones as Tokens

- Virtually all computer users have mobile phones
- Phone loss is serious for personal reasons; people guard them and notice their loss
- But—smart phones can be hacked; there are already malware apps that intercept challenge/response messages
- You're also trusting the phone company; it's not an encrypted channel

The SecurID Token

- Token (or software-based equivalent) knows time T , key k
- Device displays $E_k(T)$; user sends that
- Server matches that for $t \in [T' - \epsilon, T' + \epsilon]$
- (Reality is more complicated: server measures historical skew of token's clock and uses that to calculate T')



(Photo by Alexander Klink, via Wikimedia)

What Must the Server Know?

- For cryptographic challenge-response or the SecurID, the server must have a database of keys
- This database is obviously very sensitive—tokens are commonly tamper-resistant to keep k from the users
- We can't risk storing plaintext passwords, but we have to store plaintext keys. . .
- Oops

Provisioning k

- Where does k come from?
- More precisely, the IT department has just received a shipment of 1,000 tokens. How do $k_1, k_2, \dots, k_{1000}$ get entered into the database and associated with the proper token?
- It appears to come in a file from the vendor—so the vendor knows, or at least knew, the keys
- Lockheed was penetrated a few years ago, using data that was apparently stolen from RSA by (Chinese?) attackers

My Theory on the RSA Attack

- Assume that s is a token's serial number. Perhaps $k_i = E_{\mathcal{K}}(s_i)$ or $k_i = H(\mathcal{K}, s)$ where \mathcal{K} is a per-customer key
- Alternatively, perhaps RSA produces random k_i and stores a per-customer file of $\langle s_i, k_i \rangle$ pairs, to help customers who have lost their copy
- Either this file or Lockheed's \mathcal{K} was stolen
- Note: Users supply a login name; tokens are indexed by serial number s . How did the attackers find Lockheed login names and map these to token serial numbers?

Lost Tokens

- What do you do about lost tokens?
- What do you do about lost tokens by traveling users?
- How are they authenticated? Are they allowed to log in before the new token is shipped to them?
- Token authentication can be very secure—but in some cases, such as stolen key files or lost tokens, the problems and recovery mechanisms are just about the same as for passwords
- Authentication is a systems problem!

Cryptographic Authentication

- Authentication is a side-effect of a cryptographic negotiation to establish a session key
- Examples: Kerberos; client-side certificates in TLS
- Attractive, since you probably need the session key to protect the rest of the session
- There are limits to its security. . .

Limits of Cryptographic Authentication

- Cryptographic authentication is based on a private or secret key—how is it protected?
- Stored on the user's computer? Again, how is it protected? A password? (Guessing attacks may still be possible.)
- How do you synchronize the private key amongst multiple client devices? How is it protected during synchronization?
- You can store the key in outboard hardware (e.g., a USB widget), but that doesn't work well for mobile devices
- You still need ways to recover from lost devices and/or USB widgets
- On the other hand, if public key cryptography is used, the server's key store isn't sensitive

Biometric Authentication

- “Something you are”
- Common forms: fingerprint, iris scan, retina scan, facial recognition
- Other types: typing rhythm, voiceprint, hand geometry
- Marketing materials suggest that this is “perfectly” secure
- Not so fast. . .

Limits of Biometric Authentication

- If the server's database is stolen, can an attacker construct a fake biometric?
- 👉 Candy fingerprints and life-size pictures have fooled sensors!
- There's at least one report of a severed finger being used to start a car
Defense: *liveness detectors*
- How do you change your “password”?
- Can you use a biometric to encrypt/decrypt a key store (e.g., the **Keychain** on MacOS)?
- Nothing to forget or lose, but what about injuries, illnesses, and oddities? (About 5% of people do not have easily scannable fingerprints.)

False Accept/False Reject

- Technology is improving, though still well short of perfect
- There are always false accepts and false rejects
- In fact, there's a tradeoff—the lower the false accept rate, the higher the false reject rate
- Is the false accept rate low enough for security?
- How do you cope with false rejects?

Federated Authentication

- Log in to one site—Google, Facebook, Microsoft, others—and let it vouch for you to all other sites
- Can use strong (or weak. . .) authentication to that one site
- Is the site trustable? Do you use your company badge to get in to work, or a credit card from your bank?
- What if that site is compromised?
- There are privacy issues: this one site learns everywhere else you go
- Not very popular yet for organizational use; some uptake on the Web

Authentication as a Systems Problem

- The many different forms of authentication have a great deal in common:
 - Secondary authentication
 - Dealing with server compromise
 - Susceptibility to guessing attacks
 - Administrative infrastructure
- These pieces interact

Properties of Authentication Mechanisms

	Guessing	Forgetting	Device loss	Server file compromise	Temp access	External trust
Passwords	✗	✗	✓	✗	✓	✓
Lamport's	Maybe	✗	✗	✗	?	✓
Chall/resp	✓	✓	✗	✗✗	✗	✓
SMS	✓	✓	?	✓	✗	?
Time-based	✓	✓	✗	✗✗	?	✗
Crypto	✓	✓	?	✗, ✓	?	✓
Biometric	✓	✓	?	✗	✗	✓
Federated	?	?	✓	✓	?	✗

- ✓ No particular problem; strength of this mechanism
- ? Some trouble or implementation-dependent
- ✗ Significant risk
- ✗✗ Very serious risk

There Are No Perfect Solutions

- All mechanisms have their shortcomings
- Most of the effort thus far has focused on eliminating passwords, because of the problem of guessing
- ☞ But other schemes have different shortcomings (including cost)

Passwords Aren't Going Away

- They're simple; everyone understands them
- They're low-cost
- Well, the cost isn't that low, when you account for recovery from forgotten passwords
- Other types of authentication have their own challenges
- We have to learn to handle them properly