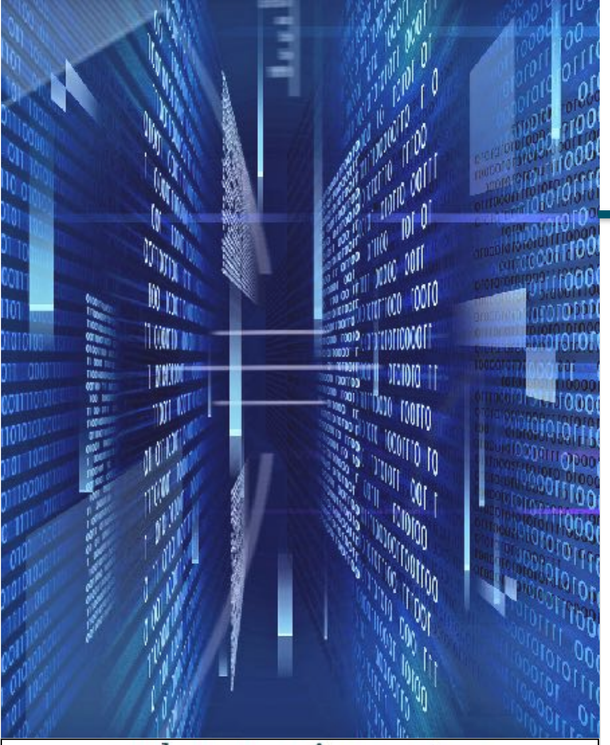


cybersecurity centre
GCHQ Academic Centre of Excellence EPSRC

Cybersecurity Southampton

CyberSecuritySoton.org [w]

@CybSecSoton [fb & tw]

UNIVERSITY OF Southampton

COMP6224 (2016)
week 5: Passwords

Vladimiro Sassone
Cyber Security Centre
University of Southampton

user authentication

three possible approaches, several techniques

- something the user ***knows***
  e.g., PIN, password, secret question, …
- something the user ***has***
  e.g., chip card, physical key, pass, …
- something the user ***is***
  e.g., fingerprint, face, voice, …

each has pros&cons; here we focus on passwords

password as authenticators

extremely common, often misused

mutually-agreed code words,
in a traditional `*challenge-response*' approach

halt! who goes there? friends! passphrase!

Potential problems

- intrinsically user-unfriendly (invent something hard to remember, but don't note it down)
- hard to use (do not reuse passwords, invent more and more, and remember them all)
- hard to manage (loss, revocation, reset, …)
- subject to sophisticated attacks (password cracking)
- subject to guessing & involuntary disclosure (social engineering)
- easy to share and reuse unduly

password systems — a word from the experts

This increase in password use is mostly due to the surge of online services, including those provided by government and the wider public sector. Passwords are an easily-implemented, low-cost security measure, with obvious attractions for managers within enterprise systems. However, this proliferation of password use, and increasingly complex password requirements, places an unrealistic demand on most users.

Inevitably, users will devise their own coping mechanisms to cope with 'password overload'. This includes writing down passwords, re-using the same password across different systems, or using simple and predictable password creation strategies. A study within a Scottish NHS trust found that 63% of its users admitted to re-using passwords.[2]

password systems — a word from the experts

This increase in password use is mostly due to the surge of online services, including those provided by government and the wider public sector. Passwords are an easily-implemented, low-cost security measure, with obvious attractions for managers within enterprise systems. However, this proliferation of password use, and increasingly complex password requirements, places an unrealistic demand on most users.

Inevitably, users will devise their own coping mechanisms to cope with 'password overload'. This includes writing down passwords, re-using the same password across different systems, or using simple and predictable password creation strategies. A study within a Scottish NHS trust found that 63% of its users admitted to re-using passwords.[2]

Approaches to discovering passwords include:

- social engineering eg phishing; coercion
- manual password guessing, perhaps using personal information 'cribs' such as name, date of birth, or pet names
- intercepting a password as it is transmitted over a network
- 'shoulder surfing', observing someone typing in their password at their desk
- installing a keylogger to intercept passwords when they are entered into a device
- searching an enterprise's IT infrastructure for electronically stored password information
- brute-force attacks; the automated guessing of large numbers of passwords until the correct one is found
- finding passwords which have been stored insecurely, such as handwritten on paper and hidden close to a device
- compromising databases containing large numbers of user passwords, then using this information to attack other systems where users have re-used these passwords

password systems — a word from the experts

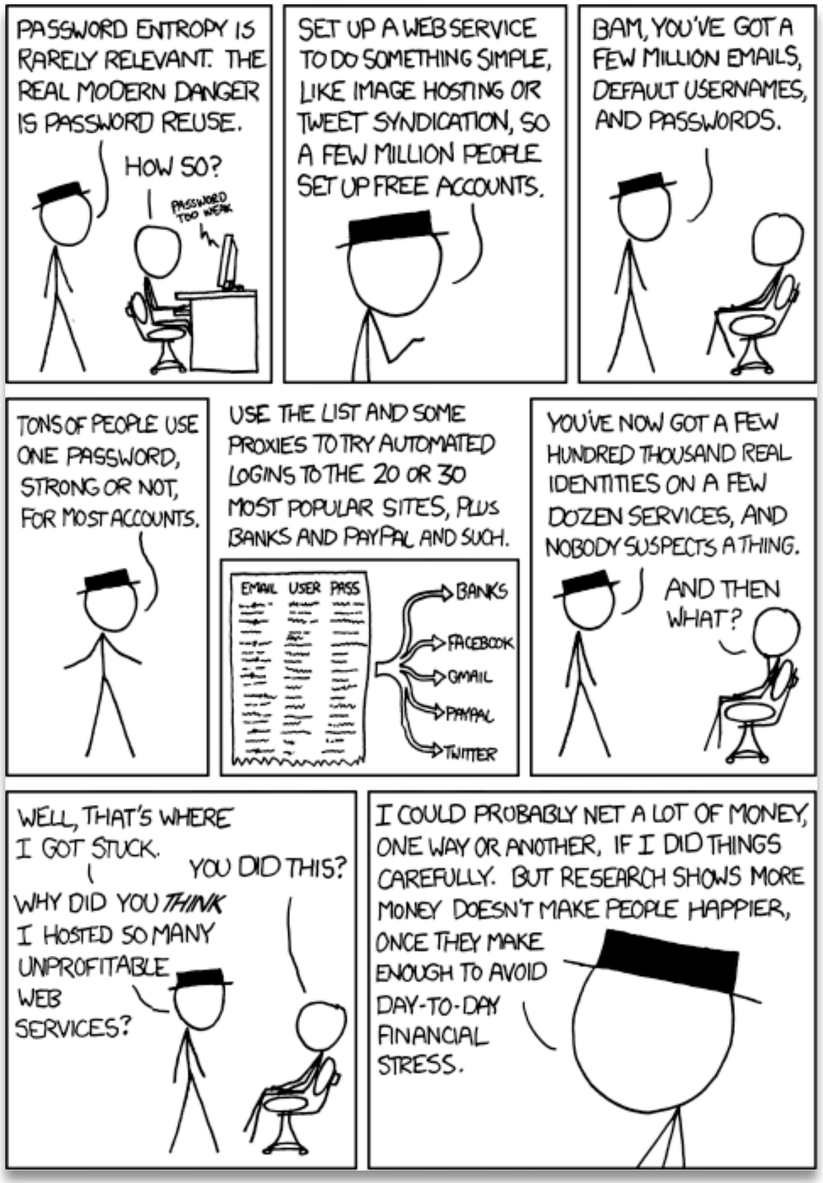
This increase in password use is mostly due to the surge of online services, including those provided by government and the wider public sector. Passwords are an easily-implemented, low-cost security measure, with obvious attractions for managers within enterprise systems. However, this proliferation of password use, and increasingly complex password requirements, places an unrealistic demand on most users.

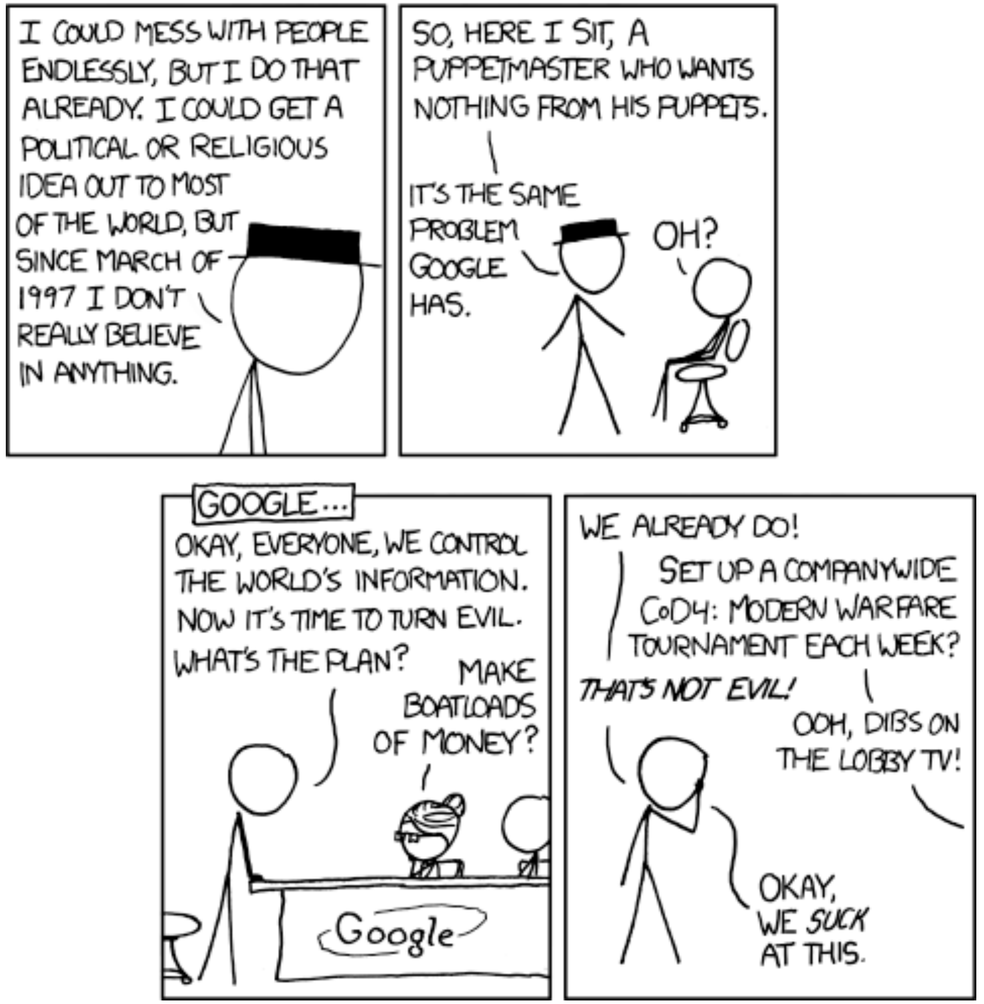
Inevitably, users will devise their own coping mechanisms to cope with 'password overload'. This includes writing down passwords, re-using the same password across different systems, or using simple and predictable password creation strategies. A study within a Scottish NHS trust found that 63% of its users admitted to re-using passwords.[2]

Approaches to discovering passwords include:

- social engineering eg phishing; coercion
- manual password guessing, perhaps using personal information 'cribs' such as name, date of birth, or pet names
- intercepting a password as it is transmitted over a network
- 'shoulder surfing', observing someone typing in their password at their desk
- installing a keylogger to intercept passwords when they are entered into a device
- searching an enterprise's IT infrastructure for electronically stored password information
- brute-force attacks; the automated guessing of large numbers of passwords until the correct one is found
- finding passwords which have been stored insecurely, such as handwritten on paper and hidden close to a device
- compromising databases containing large numbers of user passwords, then using this information to attack other systems where users have re-used these passwords

password reuse — what the issue really is…





password attacks

UNIVERSITY OF Southampton

online:

attempts to guess the password by trying it on straight on the system
easy to defend against, because slow and system can enforce back-off rules

offline:

obtain a copy of the password file and have a go at it: no limit on the number of tries

can be as fast as 700m tries/sec! (that is, test the entire English dictionary 3.5k times a sec…)

brute-force:

test each potential password at random (not very smart, takes a long time…)

dictionary:

test each potential password according to a priority list (e.g., dictionary)

bespoke:

test password more likely for the specific user (personal information cribs)

password attacks

online:

attempts to guess the password by trying it on straight on the system
easy to defend against, because slow and system can enforce back-off rules

offline:

obtain a copy of the password file and have a go at it: no limit on the number of tries

can be as fast as 700m tries/sec! (that is, test the entire English dictionary 3.5k times a sec…)

brute-force:

test each potential password at random (not very smart, takes a long time…)

dictionary:

test each potential password according to a priority list (e.g., dictionary)

powerful!

bespoke:

test password more likely for the specific user (personal information cribs)

defending against brute-force attacks

password file is the treasure chest everybody is looking for

so, first things first: never ever store password in plaintext!

30% of websites do!!

http://plaintextoffenders.com

should we encrypt?

better idea! use *one-way encryption* (aka "*hashing*", eg SHA, bcrypt, scrypt, …); indeed you never need to decrypt, and you never do (which is why a proper setup knows no passwords, just their hashes), they just compare password hashes to each other.

good, init?

not bad, but… leaves passwords open to attacks by pre-computed hashes (rainbow tables);

to force attackers to recompute hashes, we append *12 bit random number* to each password (typically a *system-wide* + a *per-password* part); this is called a *salt*.

the salt must be stored in clear with the password, and so it adds nothing to the effort of cracking a single password; but each password now has 2^12 version, which makes lookup and pre-computed dictionary attacks ineffective.

storing passwords

Passwords should never be stored as plain text, even if the information on the protected system is relatively unimportant.

We've read how users re-use passwords and employ predictable password creation strategies. This means an attacker who gains access to a database containing plain text passwords already knows a user's credentials for one system. They can use this information to attempt to access more important accounts, where further damage can be done.

Periodically search systems for password information that is stored in plain text. Consider establishing automated processes that (for example) regularly search for clear text emails and documents containing the word 'password' in the filename or body.

If you need to provide access to clear text passwords (such as when a selection of characters are requested during a login), use an alternative method of protection (for example, on-demand decryption).

In summary

- Never store passwords as plain text.
- Produce hashed representations of passwords using a unique salt for each account.
- Store passwords in a hashed format, produced using a cryptographic function capable of multiple iterations (such as SHA 256).
- Ensure you protect files containing encrypted or hashed passwords from unauthorised system or user access.
- When implementing password solutions use public standards, such as PBKDF2, which use multiple iterated hashes.

brute force "blind" attack

If we want to check all passwords of length n that consist entirely of lower-case letters, the number of such passwords is 26^n. If we suppose that the password consists of printable characters only, then the number of possible passwords is somewhat less than 95^n.

critical factor here is the amount of time it takes to salt and hash a particular password candidate and check it against the value stored in the password file

n	26 lower-case letters	36 lower-case letters and digits	62 alpha-numeric characters	95 printable characters	all 128 ASCII characters
1	30 msec.	40 msec.	80 msec.	120 msec.	160 msec.
2	800 msec.	2 sec.	5 sec.	11 sec.	20 sec.
3	22 sec.	58 sec.	5 min.	17 min.	44 min.
4	10 min.	35 min.	5 hrs.	28 hrs.	93 hrs.
5	4 hrs.	21 hrs.	318 hrs.	112 days	500 days
6	107 hrs.	760 hrs.	2.2 yrs.	29 yrs.	174 yrs.

e.g., 1.25 ms

“cracking” speed

A modern server can calculate the MD5 hash of about 330MB every second. If your users have passwords which are lowercase, alphanumeric, and 6 characters long, you can try *every single possible password of that size* in around **40 seconds**.

Rainbow tables, despite their recent popularity as a subject of blog posts, have not aged gracefully. CUDA/OpenCL implementations of password crackers can leverage the massive amount of parallelism available in GPUs, peaking at billions of candidate passwords a second. You can literally test all lowercase, alphabetic passwords which are ≤7 characters in less than 2 seconds. And you can now rent the hardware which makes this possible to the tune of less than $3/hour. For about $300/hour, you could crack around 500,000,000,000 candidate passwords a second.

If you’re willing to spend about 2,000 USD and a week or two picking up CUDA, you can put together your own little supercomputer cluster which will let you try around 700,000,000 passwords a second. And that rate you’ll be cracking those passwords at the rate of more than **one per second.**

password guessing approaches

short dictionary, complete english/non-english word list

same as the above but with capitalisation and common substitutions

very well known lists of common passwords, as e.g., password123

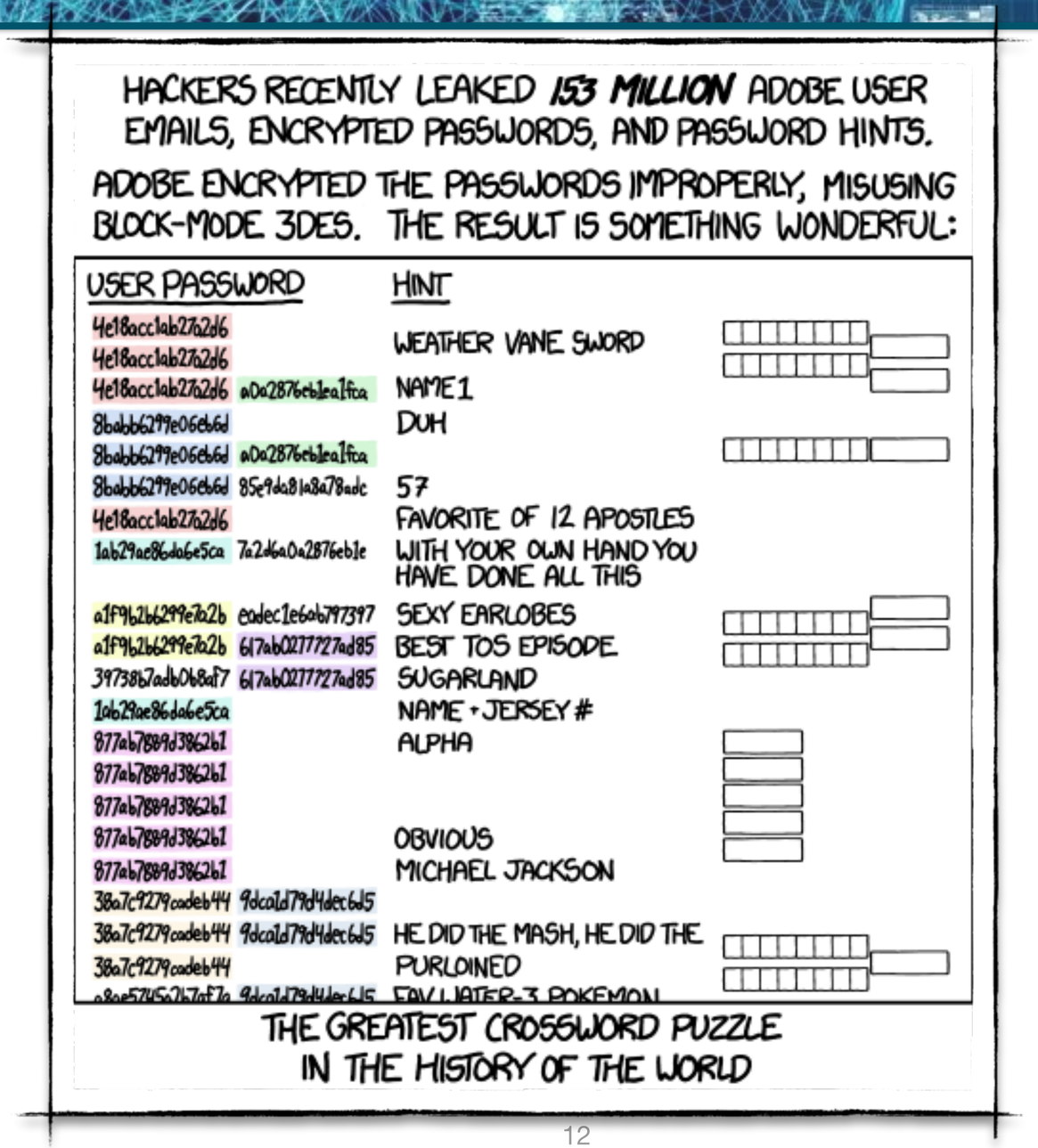
lists of names

common keyboard patterns (e.g., qwerty, azerty, …)

passwords likely for user of the system (names of self, spouse, child, brother/sister, birthdate, address, pet, home town, …)

brute force full character set

password cracking — crossword puzzles can be solved



limitations of user generated passwords

UNIVERSITY OF Southampton

User-generated password schemes are more commonly used than machine-generated ones, due to being simpler, cheaper and quicker to implement. However, user-generated password schemes carry risks that machine-generated schemes do not.

Studies of user-generated password schemes have shown that they encourage insecure behaviours. These include using the most common passwords, re-using the same password over multiple systems, and adopting predictable password generation strategies (such as replacing the letter 'o' with a zero).

Attackers are familiar with these strategies and use this knowledge to optimise their attacks. Most dictionaries for brute-force attacks will prioritise frequently used words and character substitutions. This means that systems with user-generated passwords will normally contain a large number of weak passwords that will quickly fall to an automated guessing attack.

Technical controls vs complex passwords

Traditionally, organisations impose rules on the length and complexity of passwords. However, people then tend to use predictable strategies to generate passwords, so the security benefit is marginal while the user burden is high.

The use of technical controls to defend against automated guessing attacks is far more effective than relying on users to generate (and remember) complex passwords.

For this reason, enforcing the requirement for complex character sets in passwords is not recommended. Instead, concentrate efforts on technical controls, especially:

- defending against automated guessing attacks by either using account lockout, throttling, or protective monitoring (as described in Tip 6)
- blacklisting the most common password choices

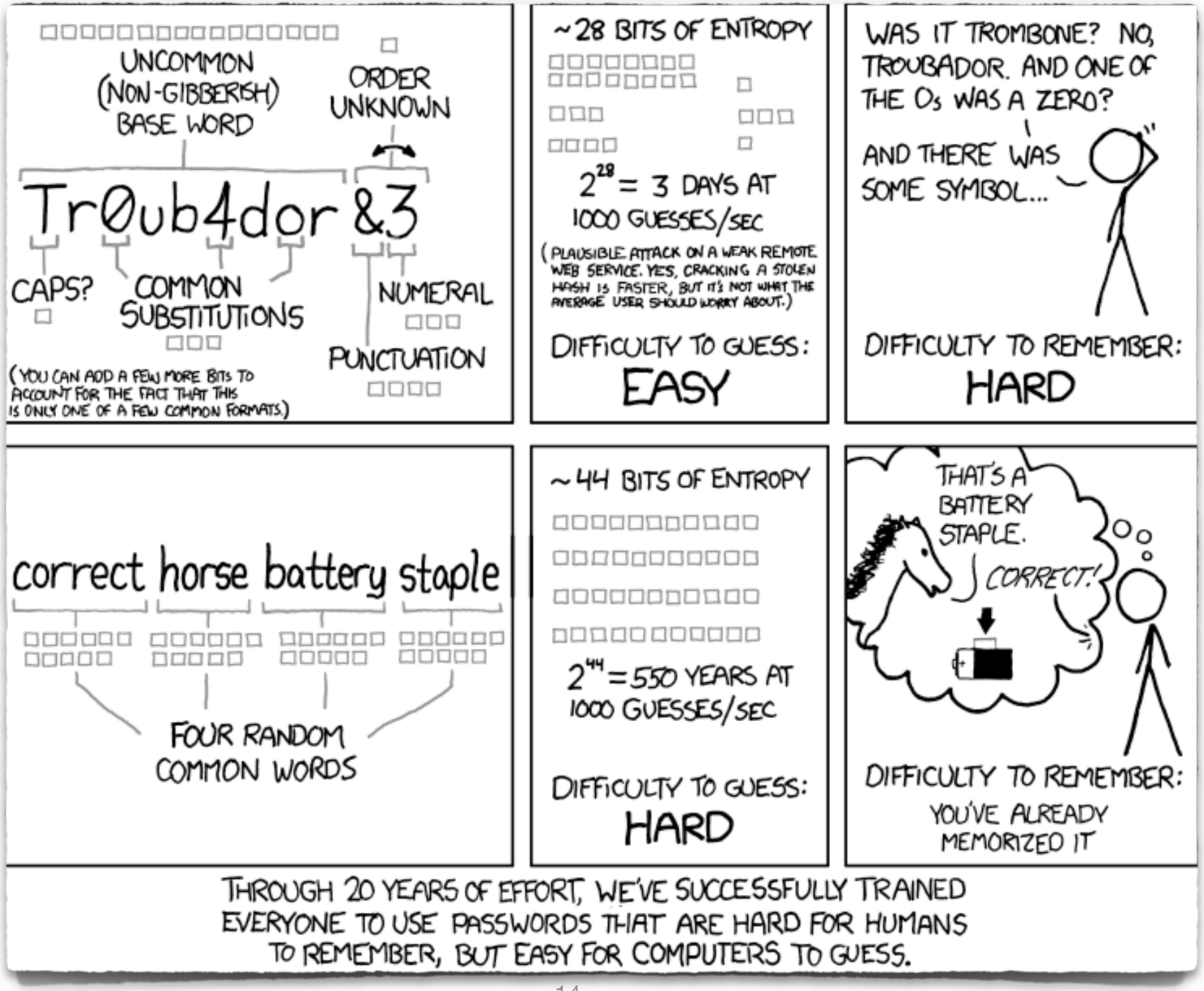
Password strength meters

Password strength meters aim to help users assess the strength of their self-generated passwords. They may steer users away from the weakest passwords, but often fail to account for the factors that can make passwords weak (such as using personal information, and repeating characters or common character strings).

In summary

- Put technical defences in place so that simpler password policies can be used.
- Reinforce policies with good user training. Steer users away from choosing predictable passwords, and prohibit the most common ones by blacklisting.
- Tell users that work passwords protect important assets; they should never re-use passwords between work and home.
- Be aware of the limitations of password strength meters.

password metrics — cannot do without attacker model



limitations of machine generated passwords

Machine-generated passwords eliminate those passwords that would be simple for an attacker to guess. They require little effort from the user to create, and, depending on the generation scheme, can produce passwords that are fairly easy to remember.

The main advantage of machine-generated schemes is that they eliminate those passwords that would be simple for an attacker to guess. They also deliver a known level of 'randomness' so it's possible to calculate the time it would take to crack the password using a brute-force attack.

Compared to user-generated schemes, there is no need to use blacklisting, and user training should be simpler. They also require little effort from the user for password creation.

Some machine generation schemes can produce passwords which are very difficult for people to remember. This increases both the demand on helpdesk for resets, and also the likelihood of insecure storage. They are not recommended.

Instead, use a generation scheme designed for high memorability (such as passphrases, 4 random dictionary words or CVC-CVC-CVC[4] style passwords). Ideally, you should give users a choice of passwords, so they can select the one they find the most memorable.

Technical controls

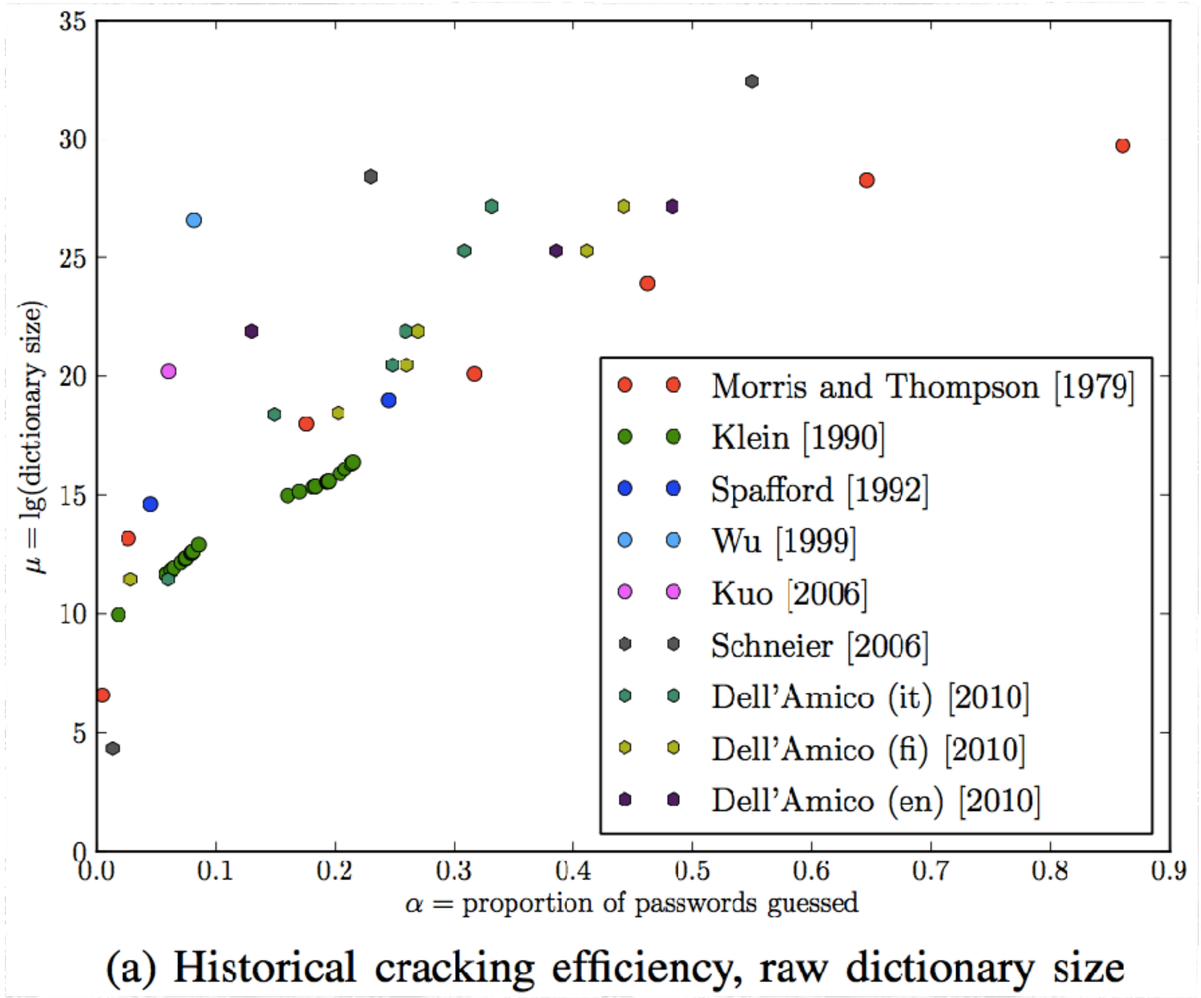
Technical controls such as account lockout, throttling or protective monitoring are still relevant when using machine-generated passwords.

In summary

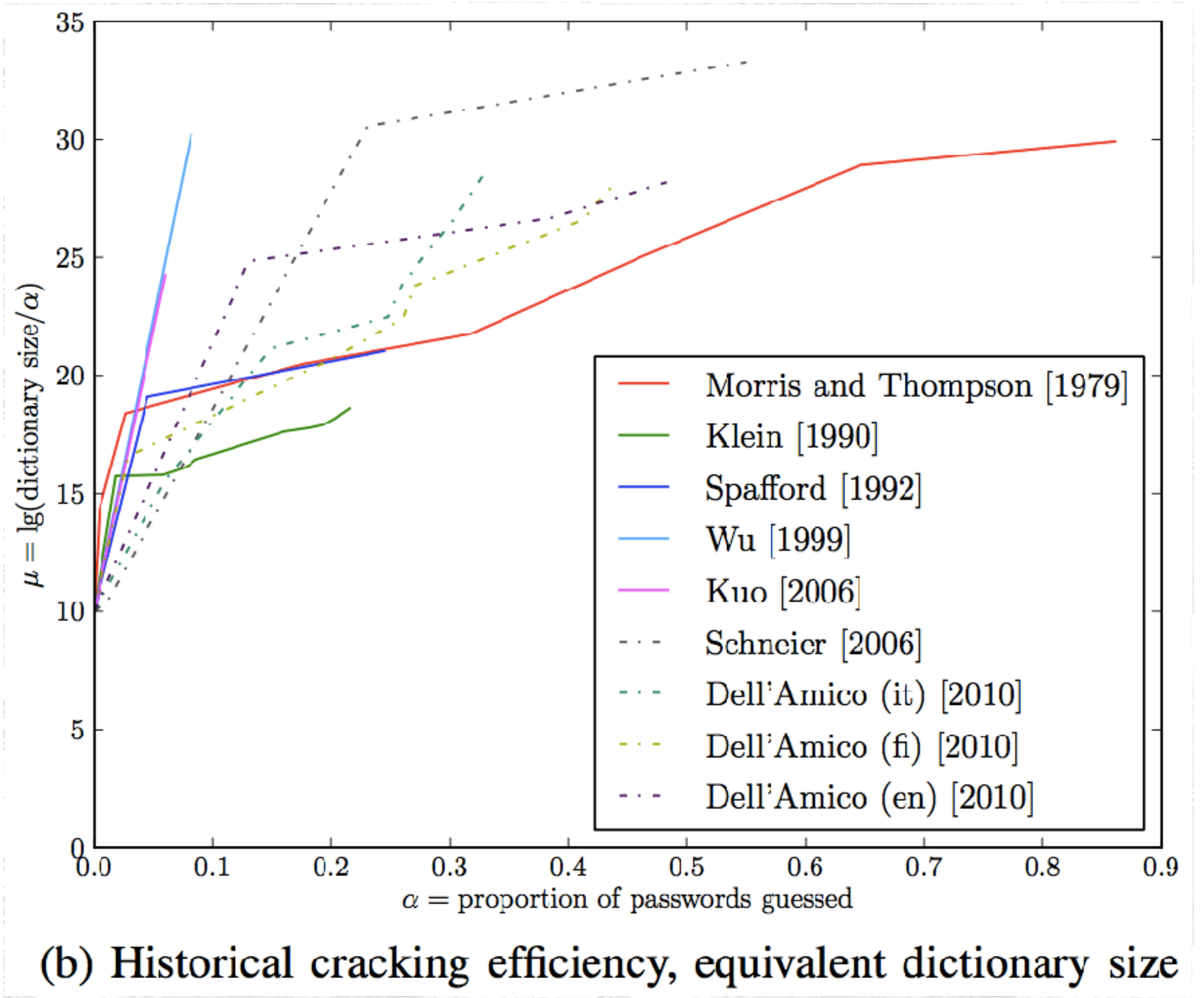
- Choose a scheme that produces passwords that are easier to remember.
- Offer a choice of passwords, so users can select one they find memorable.
- As with user-generated passwords, tell users that work passwords protect important assets; they should never re-use passwords between work and home.

CESG password advice

efficiency of dictionary attacks



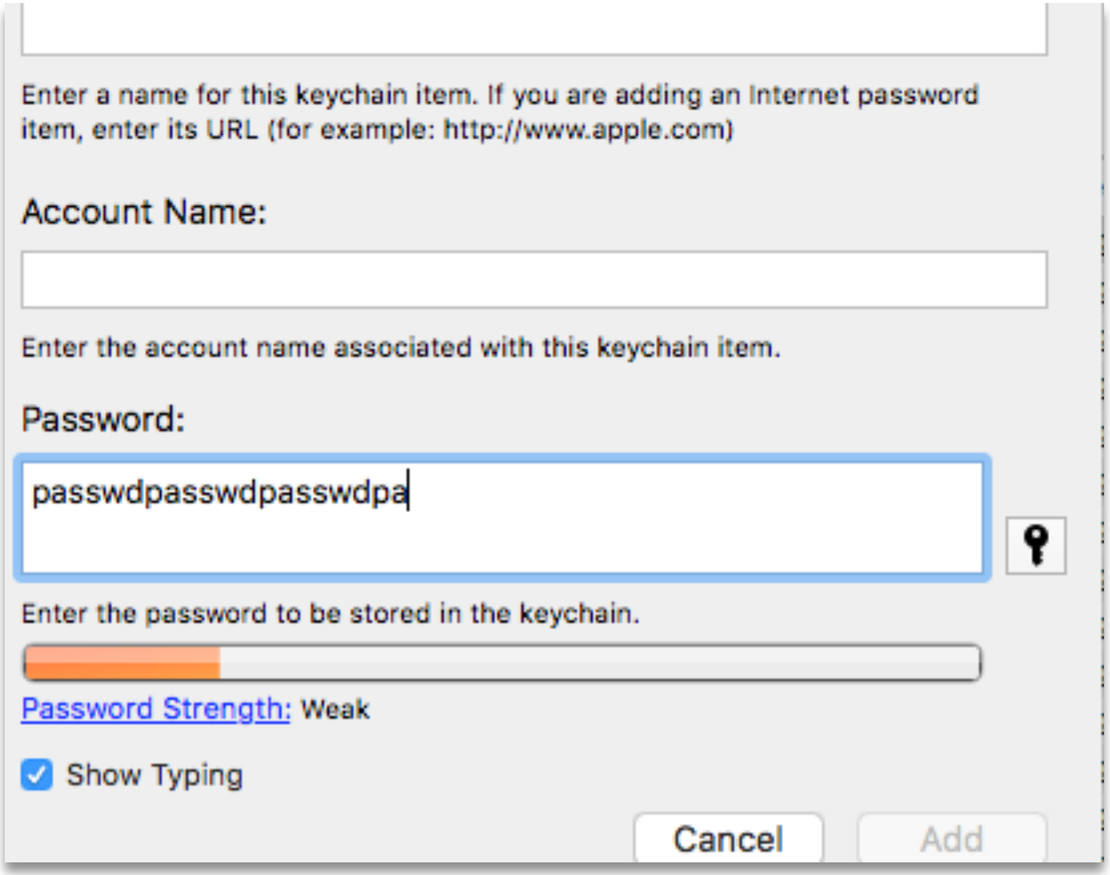
(a) Historical cracking efficiency, raw dictionary size

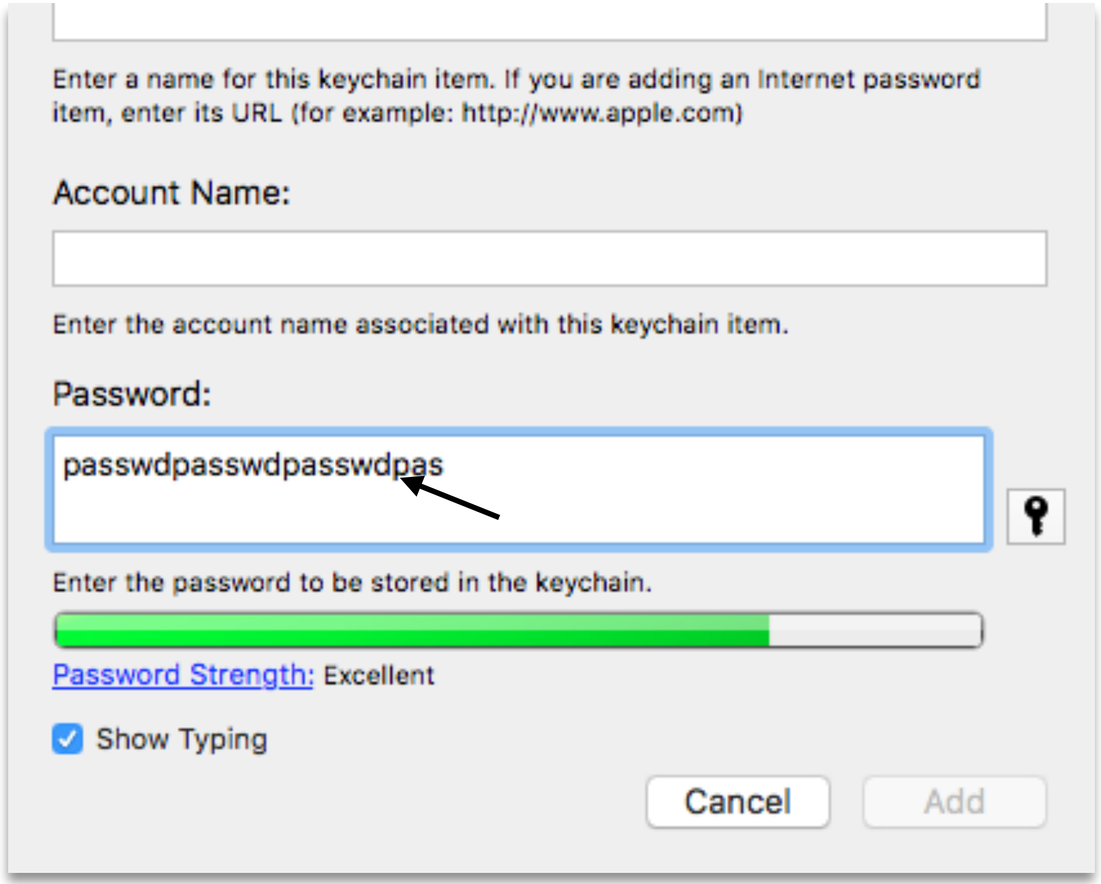


(b) Historical cracking efficiency, equivalent dictionary size

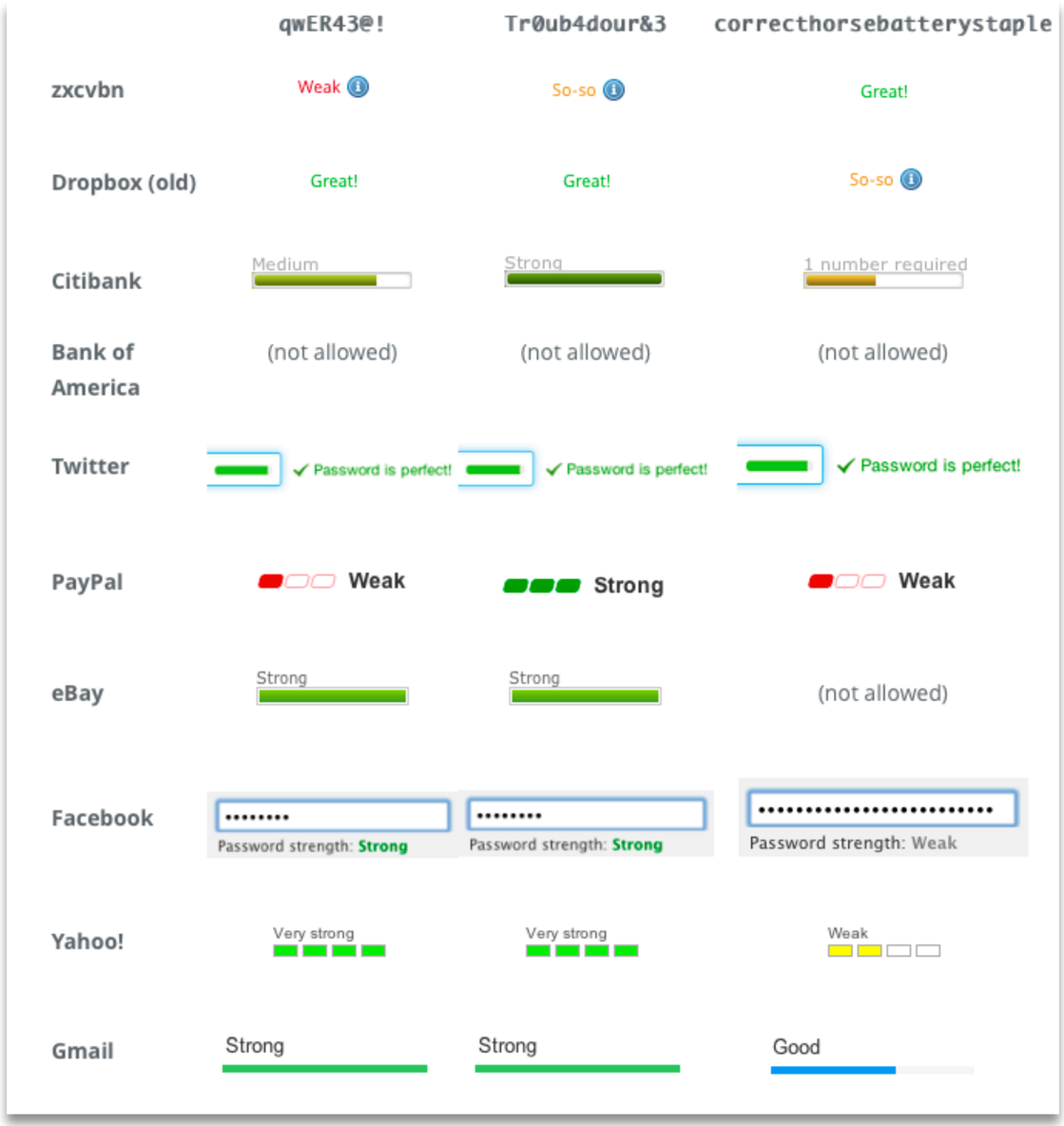
password strength — not an easy concept to measure…

Password strength is a measure of the effectiveness of a password in resisting guessing and brute-force attacks. In its usual form, it estimates how many trials an attacker who does not have direct access to the password would need, on average, to guess it correctly. The strength of a password is a function of length, complexity, and unpredictability.[1]





nor to agree on



	qwER43@!	Tr0ub4dour&3	correcthorsebatterystaple
zxcvbn	Weak	So-so	Great!
Dropbox (old)	Great!	Great!	So-so
Citibank	Medium	Strong	1 number required
Bank of America	(not allowed)	(not allowed)	(not allowed)
Twitter	✓ Password is perfect!	✓ Password is perfect!	✓ Password is perfect!
PayPal	Weak	Strong	Weak
eBay	Strong	Strong	(not allowed)
Facebook	•••••••• Password strength: Strong	•••••••• Password strength: Strong	•••••••••••••••••••••••••• Password strength: Weak
Yahoo!	Very strong	Very strong	Weak
Gmail	Strong	Strong	Good

entropy

It is usual in the computer industry to specify password strength in terms of information entropy, measured in bits, a concept from information theory. Instead of the number of guesses needed to find the password with certainty, the base-2 logarithm of that number is given, which is the number of "entropy bits" in a password. A password with, say, 42 bits of strength calculated in this way would be as strong as a string of 42 bits chosen randomly, say by a fair coin toss. Put another way, a password with 42 bits of strength would require 2^{42} attempts to exhaust all possibilities during a brute force search. Thus, adding one bit of entropy to a password doubles the number of guesses required, which makes an attacker's task twice as difficult. On average, an attacker will have to try half of the possible passwords before finding the correct one.[2]

NIST suggested entropy — historic misconception

NIST Special Publication 800-63 of June 2004 suggests the following scheme to roughly estimate the entropy of human-generated passwords:[2]

- The entropy of the first character is four bits;
- The entropy of the next seven characters are two bits per character;
- The ninth through the twentieth character has 1.5 bits of entropy per character;
- Characters 21 and above have one bit of entropy per character.
- A "bonus" of six bits is added if both upper case letters and non-alphabetic characters are used.
- A "bonus" of six bits is added for passwords of length 1 through 19 characters following an extensive dictionary check to ensure the password is not contained within a large dictionary. Passwords of 20 characters or more do not receive this bonus because it is assumed they are pass-phrases consisting of multiple dictionary words.

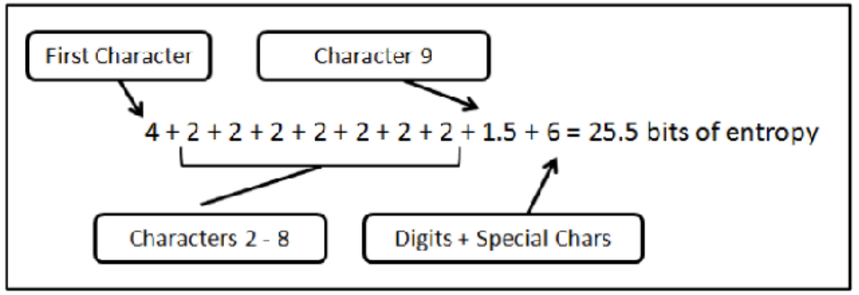


Figure 3.1: Example Calculation of NIST Entropy

Using this scheme, an eight-character human-selected password without upper case letters and non-alphabetic characters is estimated to have 18 bits of entropy. The NIST publication concedes that at the time of development, little information was available on the real world selection of passwords.

Later research into human-selected password entropy using newly available real world data has demonstrated that the NIST scheme does not provide a valid metric for entropy estimation of human-selected passwords.[12]

information theoretic attempts to measure password strength

UNIVERSITY OF Southampton

A. *Shannon entropy*

Intuitively, we may first think of the *Shannon entropy*:

$$H_1(\mathcal{X}) = \sum_{i=1}^{N} -p_i \lg p_i \qquad (1)$$

as a measure of the "uncertainty" of X to an attacker.

It has been demonstrated that H_1 is mathematically inappropriate as a measure guessing difficulty [27]–[30]. It in fact quantifies the average number of subset membership queries of the form "Is $X \in \mathcal{S}$?" for arbitrary subsets $\mathcal{S} \subseteq \mathcal{X}$ needed to identify X.[4] For an attacker who must guess individual passwords, Shannon entropy has no direct correlation to guessing difficulty.[5]

ok, but then why is the concept of entropy relevant at all?

Let X be a random variable.

This means that X represents a process that will produce values that look random to us.

Like eg throwing a coin. Or being caught sending a message in TOR. Or predicting the weather.

Or picking a given password!

The best we know about X is that it will take a certain value X = xi with prob pi.

In this context a measure of interest is the overall uncertainty of X. This is the entropy of X.

Example:

What is the entropy of X if X represents throwing a coin?

According to Shannon it is $-1/2 \log(1/2) - 1/2 \log(1/2) = 1$bit. Surprised anybody?

What is the entropy of X if the coin is biased 0.75/0.25 ?

It is $-3/4 \log(3/4) - 1/4 \log(1/4) = 0.8112\ldots$ bits. Surprised anybody?

And what if the coin had two heads?

two limit entropies — matching formulae against attackers models

1) Hartley entropy H_0: For $n = 0$, Rényi entropy is:

$$H_0 = \lg N \tag{3}$$

Introduced prior to Shannon entropy [32], H_0 measures only the size of a distribution and ignores the probabilities.

this is the simplest metric, it represents the case in which the attacker makes entirely random choices, as if every password was equally likely —in which case it works well

this is simple too, represents the attacker who has only one try, and plays in the best way she has: by trying the most likely password

2) Min-entropy H_∞: As $n \to \infty$, Rényi entropy is:

$$H_\infty = -\lg p_1 \tag{4}$$

This metric is only influenced by the probability of the most likely symbol in the distribution, hence the name. This is a useful worst-case security metric for human-chosen distributions, demonstrating security against an attacker who only guesses the most likely password before giving up. H_∞ is a lower bound for all other Rényi entropies and indeed all of the metrics we will define.

hartley entropy for common symbol sets

Symbol set	Symbol count *N*	Entropy per symbol *H*
Arabic numerals (0–9) (e.g. PIN)	10	3.322 bits
hexadecimal numerals (0–9, A-F) (e.g. WEP keys)	16	4.000 bits
Case insensitive Latin alphabet (a-z or A-Z)	26	4.700 bits
Case insensitive alphanumeric (a-z or A-Z, 0–9)	36	5.170 bits
Case sensitive Latin alphabet (a-z, A-Z)	52	5.700 bits
Case sensitive alphanumeric (a-z, A-Z, 0–9)	62	5.954 bits
All ASCII printable characters except space	94	6.555 bits
All ASCII printable characters	95	6.570 bits
All extended ASCII printable characters	218	7.768 bits
Binary (0-255 or 8 bits or 1 byte)	256	8.000 bits
Diceware word list	7776	12.925 bits

size required to meet required password entropy

Desired password entropy *H*	Arabic numerals	Hexadecimal	Case insensitive Latin alphabet	Case insensitive alphanumeric	Case sensitive Latin alphabet	Case sensitive alphanumeric	All ASCII printable characters	All extended ASCII printable characters	Diceware word list
8 bits (1 byte)	3	2	2	2	2	2	2	2	1
32 bits (4 bytes)	10	8	7	7	6	6	5	5	3
40 bits (5 bytes)	13	10	9	8	8	7	7	6	4
64 bits (8 bytes)	20	16	14	13	12	11	10	9	5
80 bits (10 bytes)	25	20	18	16	15	14	13	11	7
96 bits (12 bytes)	29	24	21	19	17	17	15	13	8
128 bits (16 bytes)	39	32	28	25	23	22	20	17	10
160 bits (20 bytes)	49	40	35	31	29	27	25	21	13
192 bits (24 bytes)	58	48	41	38	34	33	30	25	15
224 bits (28 bytes)	68	56	48	44	40	38	35	29	18
256 bits (32 bytes)	78	64	55	50	45	43	39	33	20

guessing entropy — the expected number of guesses to crack it

UNIVERSITY OF Southampton

C. Guesswork

A more applicable metric is the expected number of guesses required to find X if the attacker proceeds in optimal order, known as *guesswork* or *guessing entropy* [27], [30]:

$$G(\mathcal{X}) = E\left[\#_{\text{guesses}}(X \xleftarrow{\text{R}} \mathcal{X})\right] = \sum_{i=1}^{N} p_i \cdot i \qquad (5)$$

this represents the attacker who proceeds by trying the passwords not randomly, but from the most likely downwards. This is the best approach if you want to crack all passwords in a file.

Because G includes all probabilities in $\mathcal{X}$, it models an attacker who will exhaustively guess even exceedingly unlikely events which can produce absurd results. For example, in the RockYou data set over twenty users (more than 1 in 2^{21}) appear to use 128-bit pseudorandom hexadecimal strings as passwords. These passwords alone ensure that $G(\text{RockYou}) \geq 2^{106}$.

but attackers are not stupid…

Formally, if Eve must sequentially guess each of k passwords drawn from $\mathcal{X}$, she will need $\sim k \cdot G(\mathcal{X})$ guesses on average. However, a second guesser Mallory willing to break only $\ell < k$ of the passwords can do much better with the optimal strategy of first guessing the most likely password for all k accounts, then the second-most likely value and so on. As ℓ decreases, Mallory's efficiency increases further as the attack can omit progressively more low-probability passwords. For large values of k and ℓ, Mallory will only need to guess the most popular β passwords such that $\sum_{i=1}^{\beta} p_i \geq \alpha$, where $\alpha = \frac{\ell}{k}$. There are several possible metrics for measuring guessing in this model:

success-rate & work-factor

1) β-success-rate: A very simple metric, first formally defined by Boztaş [29], measures the expected success for an attacker limited to β guesses per account:

$$\lambda_\beta(\mathcal{X}) = \sum_{i=1}^{\beta} p_i \qquad (6)$$

fraction of the password files an attacker will crack if she only tries the β most common passwords

how many tries an optimal attacker should try per account before moving on, if she aims at cracking α% of accounts

2) α-work-factor: A related metric, first formalized by Pliam [28], evaluates the fixed number of guesses per account needed to break a desired proportion α of accounts.

$$\mu_\alpha(\mathcal{X}) = \min\left\{ j \left| \sum_{i=1}^{j} p_i \geq \alpha \right. \right\} \qquad (7)$$

3) α-guesswork: While λ_β and μ_α are closer to measuring real guessing attacks, both ignore the fact that a real attacker can stop early after successful guesses. While making up to μ_α guesses per account will enable breaking a fraction α of accounts, some will require fewer than μ_α guesses. We introduce a new metric to reflect the expected number of guesses per account to achieve a success rate α:

$$G_\alpha(\mathcal{X}) = (1 - \lambda_{\mu_\alpha}) \cdot \mu_\alpha + \sum_{i=1}^{\mu_\alpha} p_i \cdot i \qquad (8)$$

expressing β-success-rate and α-work-factor as entropy

to think of β-success-rate and α-work-factor and α-guesswork in information theoretic terms, means to *find the size k of a randomly chosen password providing the same security* (equivalent to say with Hartley entropy $H_0 = k$).

We convert each metric by calculating the logarithmic size of a discrete uniform distribution $\mathcal{U}_N$ with $p_i = \frac{1}{N}$ for all $1 \leq i \leq N$ which has the same value of the guessing metric.

For β-success-rate, since we have $\lambda_\beta(\mathcal{U}_N) = \frac{\beta}{N}$ we say that another distribution $\mathcal{X}$ is equivalent with respect to λ_β to a uniform distribution of size $N = \frac{\beta}{\lambda_\beta(\mathcal{X})}$. We take the

$$\tilde{\lambda}_\beta(\mathcal{X}) = \lg\left(\frac{\beta}{\lambda_\beta(\mathcal{X})}\right)$$

expressing α-work-factor as entropy

The conversion formula for α-work-factor is related:

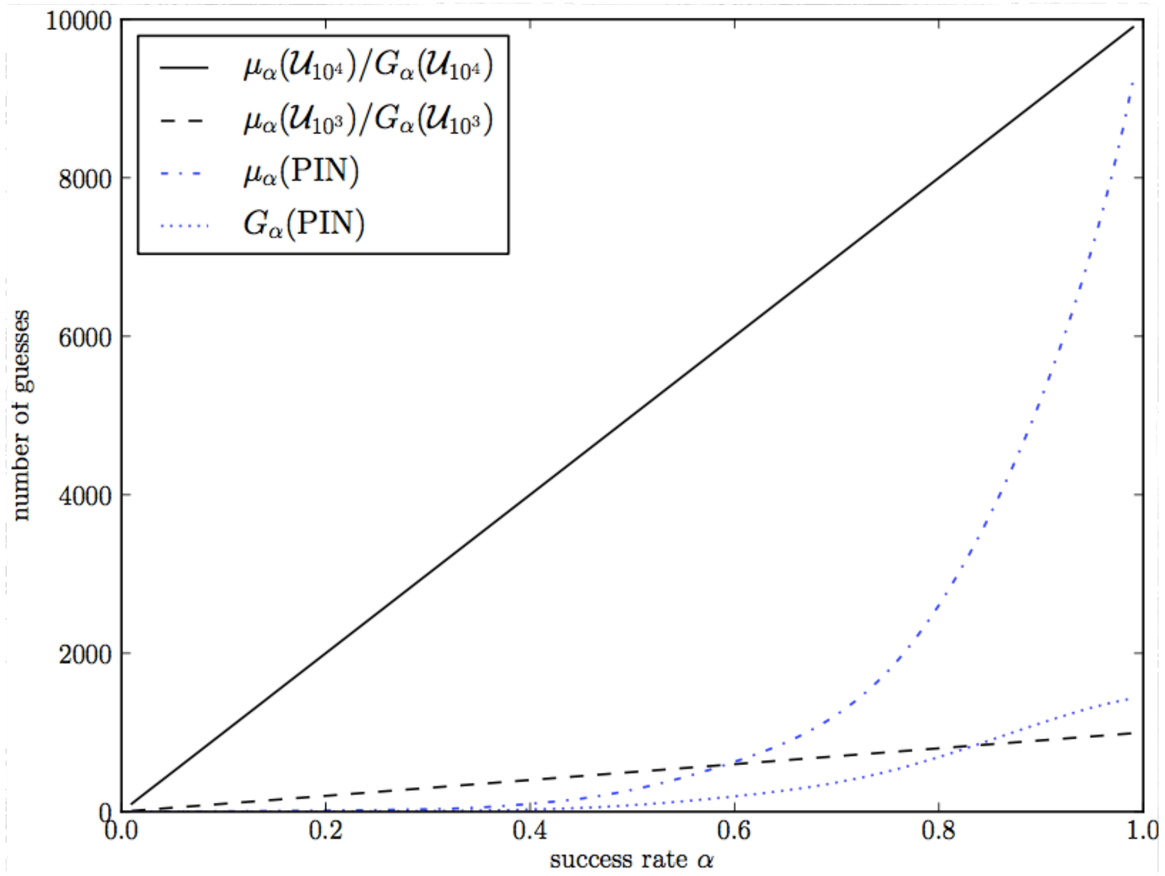
$$\tilde{\mu}_\alpha(\mathcal{X}) = \lg\left(\frac{\mu_\alpha(\mathcal{X})}{\lambda_{\mu_\alpha}}\right)$$

this is because $\mu\alpha(\mathcal{U}_N) = \alpha N$, and therefore $N = \mu\alpha(\mathcal{U}_N)/\alpha$.

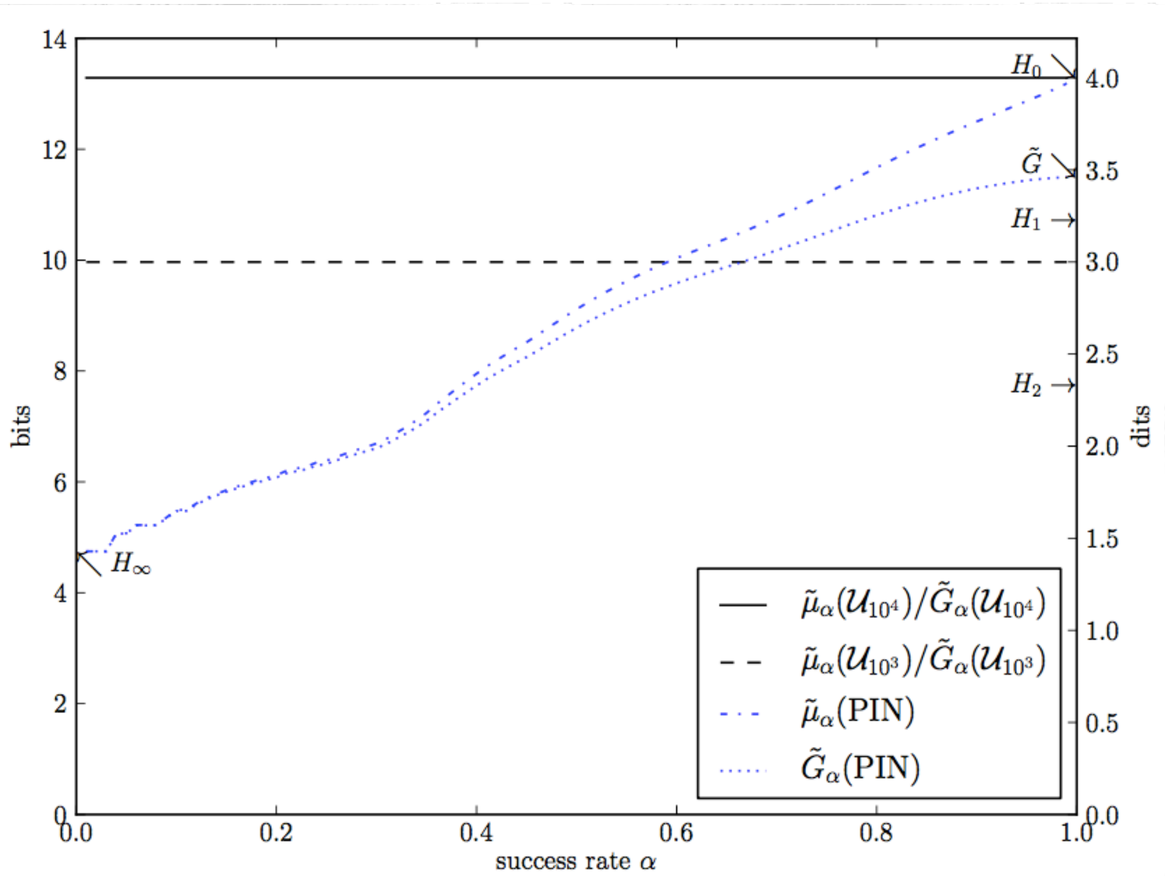
observe that λ is used instead of α, which is a clumsy step function

proceed similarly for α-guesswork

security of 4 digits PINs schemes



(a) μ_α, G_α (number of guesses)



(b) μ_α, G_α (effective key-length)

out of interest: how many bits is safe?

As a starting point, we will consider that each elementary operation implies a minimal expense of energy; Landauer's principle sets that limit at 0.0178 eV, which is $2.85*10^{-21}$ J. On the other hand, the total mass of the Solar system, if converted in its entirety to energy, would yield about $1.8*10^{47}$ J (actually that's what you would get from the mass of the Sun, according to this page, but the Sun takes the Lion's share of the total mass of the Solar system). This implies a hard limit of about $6.32*10^{68}$ elementary computations, which is about $2^{225.2}$.

Let's look at a more mundane perspective. It seems fair to assume that, with existing technology, each elementary operation must somehow imply the switching of at least one logic gate. The switching power of a single CMOS gate is about $C*V^2$ where C is the gate load capacitance, and V is the voltage at which the gate operates. As of 2011, a very high-end gate will be able to run with a voltage of 0.5 V and a load capacitance of a few femtofarads ("femto" meaning "10^{-15}"). This leads to a minimal energy consumption per operation of no less than, say, 10^{-15} J. The current total world energy consumption is around 500 EJ ($5*10^{20}$ J) per year (or so says this article). Assuming that the total energy production of the Earth is diverted to a single computation for ten years, we get a limit of $5*10^{36}$, which is close to 2^{122}.

lessons from the dawn of time — morris & thompson (i)

1. "Password security is of course only one component of overall system security, but it is an essential component."
2. "The UNIX system was first implemented with a password file that contained the actual passwords of all the users...it was completely unsatisfactory for several reasons." - decision made to encrypt passwords.
3. "... someone with access to a PDP-11 to test all lower-case alphabetic strings up to ...six characters in length. By using such a program against a collection of actual encrypted passwords, a substantial fraction of all the passwords will be found.
4. Dictionaries will find a large fraction of passwords. Good things to try are: words spelled backwards; list of first names, surnames, street & city names; the above with upper case letters; valid car registrations; room numbers; social security numbers; phone numbers; ....

lessons from the dawn of time — morris & thompson (ii)

5. Actual user choice of password: "In a collection of 3,289 passwords gathered from many users over a long period of time,

   15 were a single ASCII character;
   72 were strings of two ASCII characters;
   464 were strings of three ASCII characters;
   477 were strings of four alphamerics;
   706 were five letters, all upper-case or all lower-case;
   605 were six letters, all lower-case";
   + other "weak" passwords = 86% of total

6. "An entertaining and instructive example is the attempt made at one installation to force users to use less predictable passwords. The users did not choose their own passwords; the system supplied them randomly."

   In theory should take 112 years of exhaustive search, in practice due to a poor random number generator only 21^{15} passwords were generated

lessons from the dawn of time — morris & thompson (iii)

7. Need slower encryption
   - "DES is, by design, hard to invert, but equally valuable is the fact that it is extremely slow when implemented in software."
   - "Modern equivalent would be to use (say) 256 iterations of SHA-2 or a specialist password hashing algorithm such as bcrypt (designed to deter brute-force attacks)
8. Enforce longer passwords!
   - "If the user enters an alphabetic password ... shorter than six characters ... then the program asks him to enter a longer password."
   - "The user is warned of the risks and if he cooperates, he is very safe indeed. On the other hand, he is not prevented from using his spouse's name if he wants to."

lessons from the dawn of time — morris & thompson (iv)

9. Salted passwords
   - "when a password is first entered, the password program obtains a 12-bit random number and appends this to the password typed in by the user. The concatenated string is encrypted and both the 12-bit random quantity (called the salt) and the 64-bit result of the encryption are entered into the password file."
10. Modified DES
   - "Chips to perform the DES encryption are already commercially available and they are very fast. The use of such a chip speeds up the process of password hunting by three orders of magnitude. To avert this possibility, one of the internal tables of the DES algorithm is changed in a way that depends on the 12-bit random number."

lessons from the dawn of time — morris & thompson (v)

11. Do not tell an attacker if a username is correct!
    - "It is poor design to write the login command in such a way that it tells an interloper when he has typed in an invalid user name. The response to an invalid name should be identical to that for a valid name."
12. ...

alternatives to passwords

UNIVERSITY OF Southampton

one-time passwords

- never reused: excellent provided you don't leak the list or the method
- include in this 2-factors authentication like google authenticator

single sign-on

- user authenticated once per session on a single system which does further authentication
- strength depends on strength of original sign-on
- compromise possible by trojan horses, sniffers, wiretaps, man-in-the-middle, guessing

two-devices authentication

- like most banks do today
- compromise made possible by need to integrate services across devices and OSs

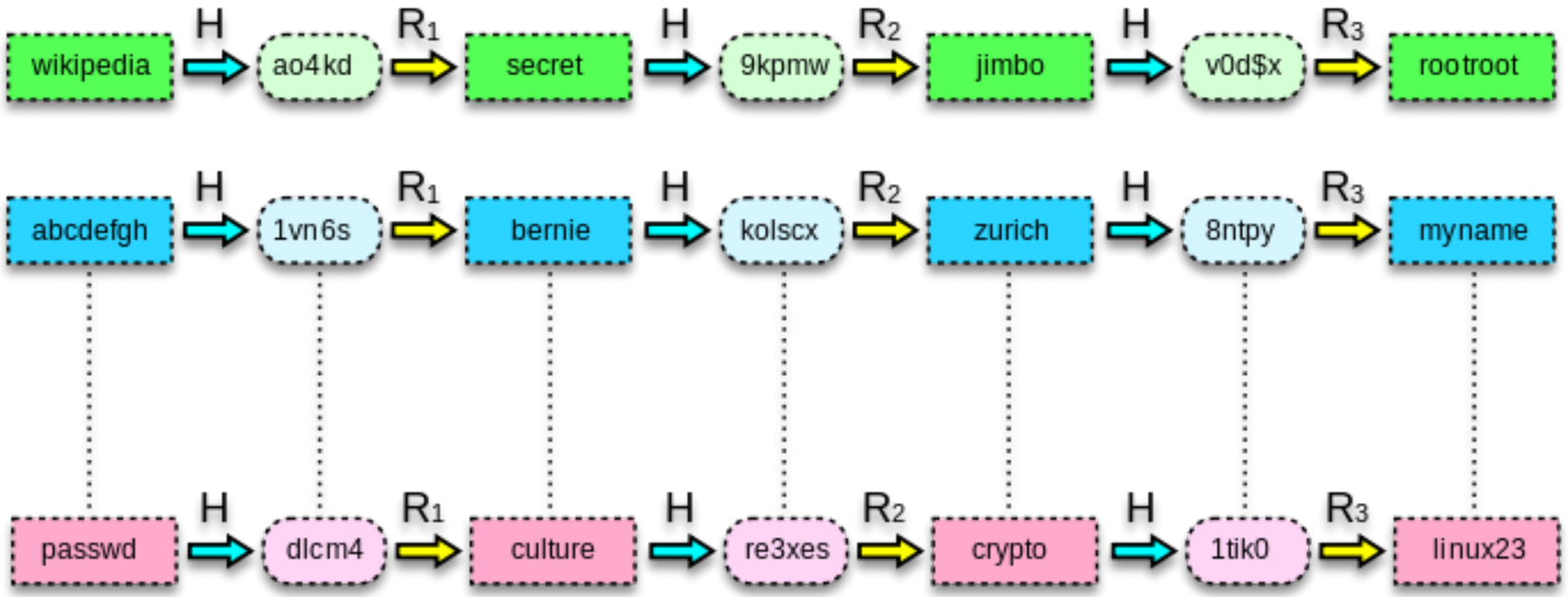
biometrics (fingerprints, voice, handwriting, …)

- not always perfect, can be spoofed, but very convenient for users… just don't leak them!!

rainbow tables

A **rainbow table** is a precomputed table for reversing cryptographic hash functions, usually for cracking password hashes.

Suppose we have a password hash function H and a finite set of passwords P. The goal is to precompute a data structure that, given any output *h* of the hash function, can either locate an element *p* in P such that H(*p*) = *h*, or determine that there is no such *p* in P. The simplest way to do this is compute H(*p*) for all *p* in P, but then storing the table requires Θ(|P|*n*) bits of space, where *n* is the size of an output of H, which is prohibitive for large |P|.



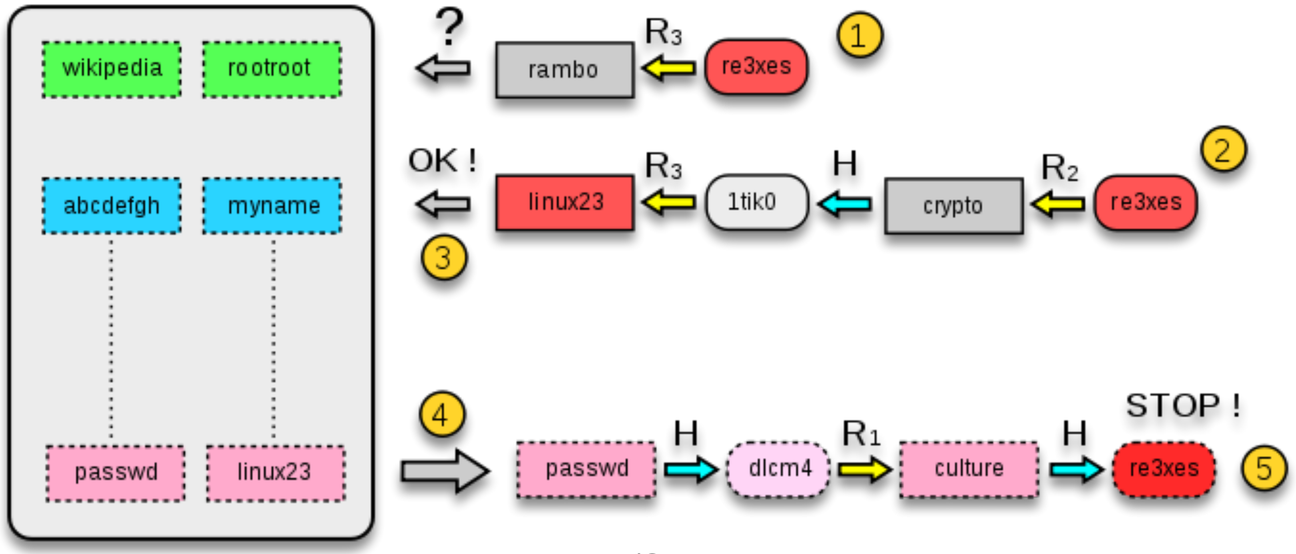
hash-chains

rainbow tables

1. Starting from the hash ("re3xes") in the image below, one computes the last reduction used in the table and checks whether the password appears in the last column of the table (step 1).
2. If the test fails (*rambo* doesn't appear in the table), one computes a chain with the two last reductions (these two reductions are represented at step 2)

   Note: If this new test fails again, one continues with 3 reductions, 4 reductions, etc. until the password is found. If no chain contains the password, then the attack has failed.

3. If this test is positive (step 3, *linux23* appears at the end of the chain and in the table), the password is retrieved at the beginning of the chain that produces *linux23*. Here we find *passwd* at the beginning of the corresponding chain stored in the table.
4. At this point (step 4), one generates a chain and compares at each iteration the hash with the target hash. The test is valid and we find the hash *re3xes* in the chain. The current password (*culture*) is the one that produced the whole chain: the attack is successful.



rainbow table

summary

passwords: an unsuitable authentication method, but arguably the “best” we have

- widely used, widely useful, but also widely misused and dangerous
- a difficult balancing act between high entropy & memorisation
- user education essential; but system admin education essential too!

brute-force, statistically-based cracking algorithms very powerful, a serious threat

- by collecting information on password real distributions from previous cracks and leaks
- by clever data structures and sophisticated algorithms

password strength difficult to measure

- no magic formulae, must take into account specific attacker model
- usefully expressed in bits to give an intuitive measure of guess-ability and work to crack

currently under active research

- no obvious replacement yet, but a lot of deep understanding and useful hybrid solutions