Computer Security and the Internet: Tools and Jewels

Chapter 3: User Authentication

P.C. van Oorschot

Sept 4, 2018

Comments, corrections, and suggestions for improvements are welcome and appreciated.
Please send by email to: paulv@scs.carleton.ca

Chapter 3

User Authentication: Passwords, Biometrics, and Alternatives

Computer users regularly enter a userid and password to access a local device or remote account. *Authentication* is the process of using supporting evidence to corroborate an asserted identity. In contrast, *identification* (*recognition*) establishes an identity from available information without an explicit identity having been asserted—such as picking out known criminals in a crowd, or finding who matches a given fingerprint; each crowd face is checked against a list of database faces for a potential match, or a given fingerprint is tested against a database of fingerprints. For identification, since the test is one-to-many, problem complexity grows with the number of potential candidates. Authentication involves a simpler one-to-one test; when an asserted userid is accompanied by a fingerprint, a single test determines if the latter matches the corresponding stored template.

Corroborating an asserted identity is sometimes the end-goal of authetication. More often, this is a sub-goal towards the end-goal of *authorization*—determining if a requested privilege or resource access should be granted to the requesting entity. For example, users are often asked to enter a password (for the account currently in use) to authorize installation or upgrading of operating system or application software.

This chapter considers *user* authentication—humans authenticating to a machine; a later chapter addresses machine-to-machine authentication and related cryptographic protocols. User authentication often relies on one or more of three things:

1. something you know (e.g., a password remembered in your head);
2. something you have (e.g., a computer or hardware token physically possessed, often holding a secret; or a device having hard-to-mimic physical properties);
3. something you are (e.g., a physical biometric, such as a fingerprint).

These three primary categories may be augmented by others, e.g., where you are (*geolocation*), and what you do (*behavioural biometrics*, and other distinguishing habits). Such factors can also be used for authorization, e.g., access to a database might be granted only to an authorized individual between 9am to 5pm.

3.1 Password authentication

Passwords provide basic user authentication. Each user authorized for a system is assigned an *account* identified by a character string called *userid*. To gain access ("log in") to their account, the user enters the userid and a *password*. This pair is transmitted to the system. The system has stored sufficient information to test if the password matches the one expected for that userid. If so, account access is granted.

A correct password does not ensure that whoever entered it is the authorized user. That would require a guarantee that no one other than the authorized user could ever possibly know, obtain, or guess the password—which is unrealistic. A correct match indicates knowledge of a fixed character string—or possibly a "lucky guess". But passwords remain useful as a (weak) means of authentication. We summarize their pros and cons later.

STORING HASHES INSTEAD OF CLEARTEXT. To verify entered userid-password pairs, the system stores sufficient information in a password file F with one row for each userid. Storing cleartext passwords p_i in F would risk directly exposing all p_i if F was stolen; system administrators and other *insiders*, including those able to access filesystem backups, would also directly have all passwords. Instead, each row of F stores a pair (*userid*, h_i), where $h_i = H(p_i)$ is a *password hash*; H is a publicly known one-way hash function.[1] The system then computes h_i from the user-entered p_i to test for a match.

PRE-COMPUTED DICTIONARY ATTACK. If password hashing alone is used as described above, an attacker may carry out the following *pre-computed dictionary attack*.

1. Construct a long list of candidate passwords, w_1, ..., w_t.
2. For each w_j, compute $h_j = H(w_j)$ and store a table T of pairs (h_j, w_j) sorted by h_j.
3. Steal the password file F containing stored values $h_i = H(p_i)$.
4. "Look up" the password p_i corresponding to a specifically *targeted* userid u_i with password hash h_i by checking if h_i appears in table T as any value h_j; if so, the accompanying w_j works as p_i. If instead the goal is to *trawl* (find passwords for arbitrary userids), sort F's rows by values h_i, then compare sorted tables F and T for matching hashes h_j and h_i (representing $H(w_j)$ and $H(p_i)$); this may yield many matching pairs, and each accompanying w_j will work as u_i's password p_i.

Exercise (Diagram of pre-computed dictionary attack). Sketch out one or more diagrams to clearly illustrate the above attack.

Exercise (Morris worm dictionary). Describe the "dictionary" used in the Morris worm incident. (Hint: see [20, 48, 51] and [49, pp.19–23].[2] Aside: this incident contributed to the rise of defensive password composition policies.)

TARGETED VS. TRAWLING SCOPE. The pre-computed attack above considered:

- a *targeted* attack specifically aimed at pre-identified users (often one); and

[1] See Chapter 2 background on cryptographic tools.

[2] The Morris worm is also discussed in Chapter 9.

- a password *trawling attack* (or *breadth-first attack*) aiming to break into any generic account, by trying many or all accounts.[3]

APPROACHES TO DEFEAT PASSWORD AUTHENTICATION. Password authentication can be defeated by several technical approaches, each targeted or trawling.

1. *Online password guessing*. Guesses are sent to the legitimate server (Section 3.2).
2. *Offline password guessing*. No per-guess online interaction is needed (Section 3.2).
3. *Password capture* attacks. An attacker intercepts or directly observes passwords by means such as: observing sticky-notes, shoulder-surfing or video-recording of entry, hardware or software keyloggers or other client-side malware, server-side interception, proxy or middleperson attacks, phishing and other social engineering, and pharming. These methods are discussed mainly in other chapters.[4]
4. *Password interface bypass*. The above three attacks are *direct attacks* on password authentication. In contrast, *bypass attacks* aim to defeat authentication mechanisms by avoiding their interfaces entirely, instead gaining unauthorized access by exploiting software vulnerabilities or design flaws (as discussed in other chapters).

Exercise (Relating attack approaches to network layout). Draw a diagram locating the above password attack approaches and instances on a network architecture diagram.

PASSWORD COMPOSITION POLICIES AND "STRENGTH". To ease remembering passwords, many users choose (rather than strings of random characters) words found in common-language dictionaries. Since guessing attacks exploit this, many systems impose *password composition policies* with rules specifying, e.g., minimum character lengths (e.g., 8 or 10), and requiring that passwords contain characters from two or three LUDS categories: lowercase (L), uppercase (U), digits (D), special characters (S). Such passwords are said to be "stronger", but this term misleads in that such increased "complexity" provides no more protection against capture attacks, and improve outcomes (whether an attack succeeds or not) against only some guessing attacks. Users also predictably modify dictionary words, e.g., to begin with a capital, and end with a digit. More accurately, such passwords have increased resilience to simple password-guessing attacks.

DISADVANTAGES OF PASSWORDS. Disadvantages multiply as the numbers of passwords that users must manage grows from just a few to tens or hundreds. Usability disadvantages include users being told, for example:

1. not to write their passwords down ("just memorize them");
2. to follow complex composition policies (with apparently arbitrary rules, some excluding commas, spaces and semi-colons while others insist on special characters);
3. not to re-use passwords across accounts;

[3] Breadth-first attacks are also discussed in Section 3.9.

[4] ***Editorial cross-check: confirm that these terms are explained in other chapters.

4. to choose each password to be *easy to remember but difficult for others to guess* (this is meaningless for users not understanding how password-guessing attacks work);

5. to change passwords every 30–90 days if password expiration policies are in use.

Beyond the above usability issues, security disadvantages as noted earlier include being vulnerable to offline guessing attacks, online guessing attacks (especially when predictably chosen, below), and capture (e.g., by malware, interception, or observation).

ADVANTAGES OF PASSWORDS. Among offsetting advantages, passwords are:

1. simple, easy to learn, and already understood by all current computer users;

2. "free" (requiring no extra hardware at the client or system/server);

3. require no extra physical device to carry;

4. allow relatively quick login, and password managers may help further (for small-keyboard mobile devices, apps may store passwords);

5. easy to change or recover if lost—electronic recovery is typically immediate with no physical travel needed or delay awaiting physical shipment (cf. Section 3.3);

6. have well understood failure modes (forgetful users learn to write them down somewhere safe);

7. require no trust in a new third party (in contrast, public key certificates require trust in organizations beyond either the client or server organization);

8. easily delegated (e.g., to a spouse or secretary while on vacation), an under-rated benefit despite the security drawback that retracting delegation is rarely done.

Passwords remain the dominant means of Internet user authentication. No alternative to date has displaced them. One might conclude that the advantages outweigh the disadvantages. But their historical position as the default authentication means has aided deployment. To displace an incumbent, often a new technology must be not just marginally, but substantially better due to inertia, universal support enjoyed by an incumbent, and interoperability with existing systems. About cost, passwords are "free" only if no costs are charged for usability or user memory. That cost is small for one password, but much larger for remembering hundreds of passwords as well as which one goes with each account.

3.2 Password guessing strategies and defenses

Password guessing attacks fall into two categories so distinct that using the term "password guessing" for both can be more confusing than helpful. This will be clear shortly.

ONLINE PASSWORD GUESSING. An *online guessing* attack can be mounted against any publicly reachable password-protected server. A human attacker or automated program is assumed to know one or more valid userids; these are typically easily obtained.

Userid-password pairs, with password guesses, are submitted sequentially to the legitimate server, which conveniently indicates if the attempt is correct or not—access is granted or denied. An obvious defensive tactic, for sites that care at all about security, is to rate-limit or *throttle* guesses across fixed time windows—for example, enforcing a hard limit on the number of incorrect login attempts per account. This may "lock out" legitimate users whose accounts are attacked,[5] a drawback that can be ameliorated by account recovery methods (Section 3.3). A variation is to increase delays, e.g., doubling system response time after successive incorrect login: 1s, 2s, 4s, and so on.

OFFLINE PASSWORD GUESSING. In *offline guessing* it is assumed that an attacker has somehow stolen a copy of the system's password hash file (as in Section 3.1's precomputed dictionary attack). While in practice this has indeed happened in many cases, it is nonetheless a large assumption not required for online guessing. (The Unix password file */etc/passwd*, historically "world readable" (Chapter 5), is rarely so today.) The hash file provides *verifiable text*, i.e., data allowing a test of correctness of password guesses without contacting the legitimate server. Consequently, the number of offline guesses that can be made over a fixed time period is limited only by the computational resources that an attacker can harness; in contrast for online guessing, even without rate-limiting, the number of guesses is limited by the online server's computing and bandwidth capacity.

ITERATED HASHING (PASSWORD STRETCHING). Offline password guessing attacks can be slowed down using a tactic called *iterated hashing* (or *password stretching*). Ideally this defence is combined with salting (below). The idea is that after hashing a password once with hash function H, rather than storing $H(p_i)$, the result is itself hashed again, continuing likewise d times, finally storing the d-fold hash $H(...H(H(p_i))...)$, denoted $H^d(p_i)$. This increases the hashing time by a factor of d, for both the legitimate server (typically once per login) and each attacker guess. Practical values of d are limited by the constraint that the legitimate server must also compute the iterated hash. A value $d = 1000$ slows attacks by a factor of 1000, and d can be adjusted upward as computing power increases, e.g., due to advances in hardware speeds.

SPECIALIZED PASSWORD HASHING FUNCTIONS. General crypto hash functions H from the 1990s like MD5 and SHA-1 were designed to run as fast as possible. This also helps offline guessing attacks, wherein hash function computation is the main work; relatively modest custom processors can exceed billions of MD5 hashes per second. As attackers improved offline guessing attacks by leveraging tools such as Graphics Processing Units (GPUs), parallel computation, and FPGAs, the idea of specialized password hashing functions to slow down such attacks arose. This led to the international Password Hashing Competition (PHC, 2013-2015), with winner Argon2 now preferred; prior algorithms were bcrypt and scrypt. Specialized hash functions used to derive encryption keys from passwords are also called *key derivation functions* (KDFs). As an older example, PBKDF2 (password-based KDF #2) takes as inputs (p_i, s_i, d, L)—a password, salt, iteration count, and desired bitlength for the resulting output to be used as a crypto key.

[5] The Pinkas-Sander protocol (Section 3.8) avoids this denial of service (DoS) problem.

Example *(GPU hashing).* GPUs are particularly well-suited for hash functions like MD5 and SHA-1, with high cost-performance due to many inexpensive custom cores. For example, the circa-2012 Telsa C2070 GPU has 14 streaming multiprocessors (SMs), each with 32 computing cores, for 448 cores in one GPU. Common machines have 4 such GPUs. Thus password hashing functions are now designed to be "GPU-unfriendly".

PASSWORD SALTING. To combat dictionary attacks (above), common practice is to *salt* passwords before hashing. Rather than storing $h_i = H(p_i)$ for userid u_i, on password registration the system selects, e.g., for $t \geq 64$, for each password a random t-bit value s_i as *salt*, and stores $(u_i, s_i, H(p_i, s_i))$; here p_i and s_i are concatenated before hashing. Thus the password is altered by the salt in a deterministic way before hashing, with s_i stored cleartext in the record to enable verification. For trawling attacks, the above dictionary attack using a pre-computed table is now harder by a factor 2^t in both computation (work) and storage—a table entry is needed for each possible value s_i. For attacks on a targeted userid, if the salt value s_i is available to an insider or read from a stolen file F, the salt does not increase the time-cost of an "on-the-fly" attack where candidate passwords are hashed in real-time. Such attacks, often still called *dictionary attacks*, no longer use massive pre-computed tables of hashes (Section 3.1). Aside: password hashing is more common than reversible encryption, which requires somehow protecting the encryption key itself.

A bonus of salting is that two users who happen to choose the same password, will almost certainly have different password hashes in the system hash file. A salt value s_i may also combine a global system salt, and a user-specific salt (including, e.g., the userid).

PEPPER (SECRET SALT). A *secret salt* (sometimes called *pepper*) is like a regular salt, but not stored. The motivation is to slow down attacks, by a method different than iterated hashing but with similar effect. When user u_i selects a new password p_i, the system chooses a random value r_i, $1 \leq r_i \leq R$; stores the secret-salted hash $H(p_i, r_i)$; and then erases r_i. To later verify a password for account u_i, the system sequentially tries all values $r^* = r_i$ in a deterministic order (e.g., sequentially, starting at a random value in $[1, R]$, wrapping around from R to 1). For each r^* it computes $H(p_i, r^*)$ and tests for a match with the stored value $H(p_i, r_i)$. For a correct p_i, one expects a successful match on average (i.e., with 50% probability) after testing half the values r^*, so if R is 20 bits, one expects on average a slow-down by a factor 2^{19}. Pepper can be combined with regular salt as $H(p_i, s_i, r_i)$, and with iterated hashing. (Aside: If the values r^* are tested beginning at a fixed point such as 0, timing data might leak information about the value of r_i.)

SYSTEM-ASSIGNED PASSWORDS AND BRUTE-FORCE GUESSING. Early government systems used system-assigned passwords.[6] The difficulty of guessing passwords is maximized by selecting each password character randomly and independently. An n-character password chosen from an alphabet of b characters then results in b^n possible passwords, i.e., a *password space* of size b^n. On such systems, there is no guessing strategy better than *brute-force guessing*: simply guessing sequentially using any enumeration (complete listing in any order) of the password space. The probability of success is

[6] For example, in 1985, FIPS 112 [43] noted that many user-chosen passwords are easily guessed, and therefore all passwords should be system-generated.

100% after b^n guesses, with success expected on average (i.e., with 50% probability) after $b^n/2$ guesses. If the passwords need not be a full n characters, a common attack strategy would first try all 1-character passwords, then all 2-character passwords, and so on. System-assigned passwords are little-used today. Their higher security comes with poor usability—humans are unable to manually remember large numbers of random strings for different accounts, even without password expiration policies (below). Random passwords are more plausible when password manager tools are used (Section 3.6).

Exercise (Password expiration/aging). Password expiration policies require users to change passwords regularly, e.g., every 30 or 90 days. Do such policies improve security? List what types of attacks they stop, and fail to stop. (Hint: see [12] and [57].)

PROBABILITY OF GUESSING SUCCESS. Simple formulas giving the probability system-assigned passwords are guessed can help inform us how to measure vulnerability. (The probability increases dramatically for user-chosen passwords, but the formulas can be modified using partial guessing metrics such as β-success rate, Section 3.9.) The baseline idea is to consider the probability that a password is guessed over a fixed period (e.g., 3 months or one year). Some government standards might dictate, for example, a maximum guessing probability of 1 in 2^{10} for Level 1 applications (low security), and 1 in 2^{20} for Level 2 applications (higher security). The parameters of interest are:

- G, the number of guesses the attacker can make per unit time;
- T, the number of units of time per guessing period under consideration;
- $R = b^n$, the size of the password space (naive case of equi-probable passwords).

Here b is the number of characters in the password alphabet, and n is the password length. Assume that password guesses can be verified by online or offline attack. Then the probability q that the password is guessed by the end of the period equals the proportion of the password space that can be covered. If $GT > R$ then $q = 1.0$, and otherwise

$$q = GT/R \text{ for } GT \leq R \tag{3.1}$$

Passwords might be changed at the end of a period, e.g., due to password expiration policies (below). If new passwords are independent of the old, the guessing probability per period is independent, but cumulatively increases with the number of guessing periods.

Example *(Offline guessing).* For concreteness, consider T = 1 year (3.154 x 10^7s); truly random system-assigned passwords of length $n = 10$ from an alphabet of $b = 95$ printable characters yielding $R = b^n = 95^{10} = 6 \times 10^{19}$; and $G = 100$ billion guesses per second (this would model an attack with a relatively modest number of modern GPUs, but otherwise favourable offline attack conditions assuming a password hash file obtained, a fast hash function like MD5, and neither iterated hashing nor secret salts). A modeling condition favouring the defender is the assumption of system-assigned passwords, which have immensely better guess-resistance than user-chosen passwords (below). Given these model strengths and weaknesses, what do the numbers reveal?

$$q = GT/R = (10^{11})(3.154 \times 10^7)/6(10^{19}) = 0.05257 \tag{3.2}$$

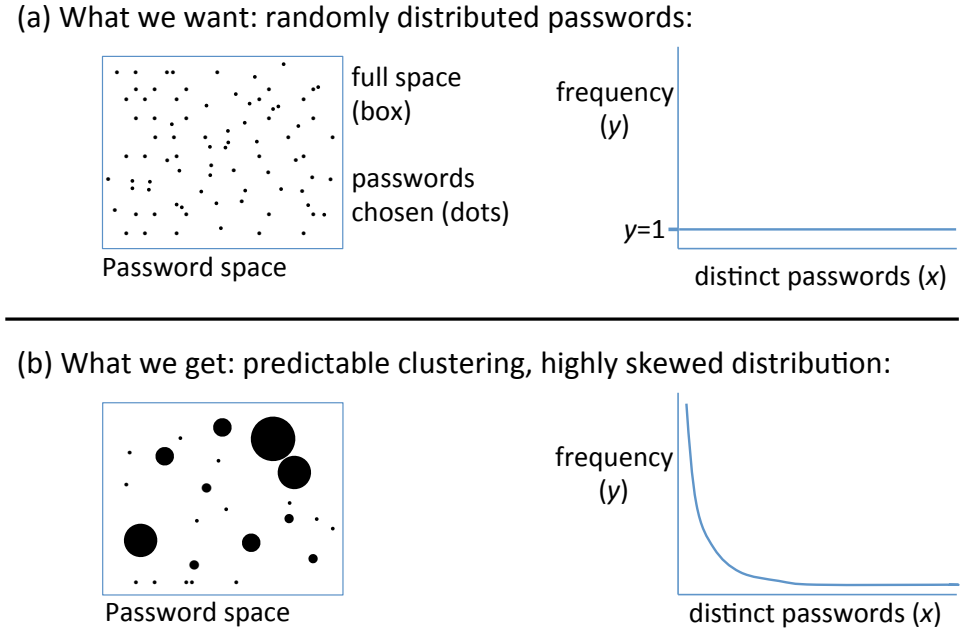

Figure 3.1: Password distributions (illustrative). Ideally, chosen passwords are unique ($y = 1$) with most unchosen ($y = 0$). Diameter represents frequency a password is chosen.

Oops! A success probability over 5% far exceeds both 2^{-10} and 2^{-20} from above. These conditions are too favourable for an attacker; a better defensive stance is needed.

Equation 3.1 can be rearranged to dictate a lower bound on password length, if other parameters are fixed. For example, if security policy specifies an upper bound on probability q, for a fixed guessing period T and password alphabet of b characters, we can determine the value n required (from $R = b^n$) if we have confidence in an estimate for G, since from $R = b^n$ and Equation 3.1 we have:

$$n = \lg(R)/\lg(b) \text{ where } R = GT/q. \tag{3.3}$$

Alternatively, to model an online attack, Equation 3.1 can determine what rate-limiting suffices for a desired q, from $G = qR/T$.

USER-CHOSEN PASSWORDS AND SKEWED DISTRIBUTIONS. Many systems today allow user-chosen passwords, constrained by password composition policies and (as discussed below) password blacklists and other heuristics. Studies show that the distribution of user-chosen passwords is highly skewed: some passwords are much more popular than others. Fig.3.1 illustrates this situation. Attackers tailor their guessing strategies by trying more popular (higher estimated probability) passwords first. While originally the term *dictionary attack* loosely implied leveraging words in common-language dictionaries, it now often refers to using ordered lists of password candidates established by heuristic means (e.g., empirical studies of large password databases including huge lists published after compromises), often with on-the-fly computation of hashes (cf. p.7).

BLACKLISTING PASSWORDS AND PRO-ACTIVE PASSWORD CHECKING. Due to the phenomenon of skewed password distributions, another simple defence against (especially online) password guessing attacks is *blacklisting of passwords*. This involves composing lists of the most-popular passwords, e.g., observed from available password distributions publicly available or within an enterprise organization. These lists need not be very long—e.g., as short as thousands, or tens of thousands of passwords. An older practice originally called *pro-active password checking* is to disallow passwords at the time users originally try to register them, by looking them up to see if they are in a fixed list such as a modified dictionary. A related idea is for the system to try to "crack" its own users' passwords using only the password hashes and background computing cycles, in variations of dictionary attacks described earlier; account owners of cracked passwords are sent email notices to change their passwords because they were too easily found.

Exercise (Heuristic password-cracking tools). Look up and experiment with common password-cracking tools such as JohnTheRipper and oclHashcat. Explain what *mangling rules* are. (Hint: see [52].)

LOGIN PASSWORDS VS. PASSWORD-DERIVED ENCRYPTION KEYS. Recalling KDFs (above), passwords are sometimes used to derive cryptographic keys, e.g., for file encryption. Such password-derived keys are subject to offline guessing attacks and require high guessing resistance; options available to stop offline guessing attacks against servers such as server-side throttling are often unavailable. For web login passwords, complex composition policies are arguably a poor choice, imposing usability burdens without necessarily improving security outcomes; alternatives such as throttling, salting, iterated hashing, and blacklisting appear preferable.[7] In contrast, for password-derived keys (and master passwords for client-based password managers), complex passwords are prudent; memorability may be aided by *passphrases*, using the first letters of words in relatively long sentences, and storing written passwords in a safe place.

Example *(Password management per NIST SP 800-63B).* U.S. password guidelines were substantially revised in 2017. They include: mandating use of password blacklisting to rule out common, highly predictable, or previously compromised passwords; mandating rate-limiting to throttle online guessing; recommending against composition rules, e.g., required combinations of lowercase, uppercase, digits and special characters; recommending against password expiration, but mandating password change upon evidence of compromise; mandating secure password storage methods (salt of at least 32 bits, hashing, suitable hash iteration counts, e.g., cost-equivalent to 10,000 iterations for PBKDF2); recommending a further secret key hash (MAC) and if so mandating that the key be stored separately (e.g., in a hardware security module/HSM).

Exercise (Password guidelines—other governments). Compare the above-noted recent revisions of U.S. password guidelines [45, Part B] to updates from governments of (i) U.K., (ii) Canada, (iii) Germany, and (iv) Australia.

[7] ***Summarize defenses table [22]. Give example gap numbers, perhaps in an exercise.

3.3 Password recovery, secret questions and SMS codes

Password-based authentication inevitably leads to forgotten passwords. Since not all users write them down in a safe place for retrieval, some means of *password recovery* is essential. Usually existing passwords are not literally "recovered", because best practice avoids storing cleartext passwords at servers (see above). Rather, what is typically recovered is access to password-protected accounts, by some *password reset* method.

RECOVERY PASSWORDS AND RECOVERY LINKS. A common reset method is to send to users, using a recovery email address set up during registration, a temporary password or web page link that serves as a *capability*. On following the link or entering the temporary code, the user is prompted to immediately create a new password. Here for obvious reasons, registering the new password does not require authorization by entering an existing password (as typically required for changing passwords); the temporary capability plays that role.

LOSS OF PRIMARY EMAIL PASSWORD (CODES SENT TO TELECOM DEVICE). A complication arises if the forgotten password is for a primary email account itself. One solution is to register a secondary email address and proceed as above. An alternative is to pre-register an independent device or channel, most commonly by a phone number (mobile, or wireline with text-to-voice conversion) to which a one-time recovery code is sent by SMS/text message. Note: a compromised primary email account may be used to compromise all other accounts that use that email address for password recovery.

QUESTION-BASED RECOVERY. Another account-recovery method to address forgotten passwords is *secret questions*, also called *challenge questions* or *fallback questions*. (The term *personal knowledge questions* is also used, but risks confusion with a type of targeted attack relying on personal knowledge.) *Secret questions* are not literally questions that are secret; rather, the idea is to use secrets cued by questions. On account registration, a user provides answers to a selected subset of questions. On later forgetting a password, correctly answering questions allows the user to re-establish a new account password.

The idea is that using questions to cue information already in a user's long-term memory are easier to remember, and thus have better memorability than passwords.[8] However, recovery by secret questions fails surprisingly often in practice, including because:

1. recovery may be long-removed in time from when answers are set;
2. answers may be non-unique or change over time (e.g., favourite movie);
3. some users register false answers but forget this, or the false answers themselves.[9]

On security, secret questions are at best "weak" secrets—answer spaces small (pro baseball team, spouse's first name) or highly skewed by popularity (favourite food/city) make statistical guessing attacks easy; for targeted attacks, some answers are easily looked up online (high school attended). User-created questions, as allowed in some systems, are

[8] See also discussion of *cued recall* under graphical passwords (Section 3.7).

[9] Some users believe false answers improve security; empirical studies have found the opposite.

notoriously bad (e.g., favourite color). Salvaging security by requiring answers to several questions is at cost of reduced efficiency and increased rejection of legitimate users due to wrong answers. A big problem remains: the answers are often not secrets at all.

Thus secret questions are poor both in security (easier to attack than user-chosen passwords) and reliability (recovery fails more often than for alternatives). Furthermore, server break-in compromises secret answers if stored plaintext (rather than hashed and so on, per best practice for passwords). The result is large companies apologizing for an "entirely unforeseeable" situation, then asking you to change your mother's maiden name—within their own system, and every other system where you have used that question. This may however be easier than changing your fingerprints when biometric data (below) is similarly compromised. All of a sudden, regular passwords start to look not so bad, despite all their flaws!

STRENGTH OF RECOVERY CHANNEL. Recovery channels present new attack vectors; a minor mitigation is to limit the validity period of recovery codes or links. As a standard security principle, a recovery channel should be designed to be at least as strong as the primary authentication means—it should not introduce a new weakest link. For example, if recovery questions are used, they should not be easier to correctly guess than primary passwords. Since they usually are weaker, it is now recognized that secret questions should be used only in combination with other authenticators.

Example *(Password Reset Attack).* Password reset processes relying on secret questions or SMS codes may be vulnerable to an interleaving attack. The attacker requires control of their own web site (site-A), and a way to get users (victims) to visit, but need not intercept Internet or phone messages. Site-A encourages download of free resources/files, requiring users register first. The registration solicits information like email address (if the attack goal is to overtake that email address), or SMS/text phone contact details (see below); the latter may be solicited ("for confirmation codes") as the user attempts to download a resource. The user visits site-A and creates a new account as noted. In parallel, the attacker requests password reset on an account of the same victim at a service provider site-B (often the userid is the same email address)—an email provider or non-email account. Site-B, as part of its reset process, asks secret questions before allowing password reset; these are forwarded by the attack program to the victim on site-A, positioned as part of site-A's registration. Answers are forwarded to site-B and authorize the attacker to set a new account password on site-B. If site-B's reset process involves sending one-time codes to the contact number on record, the attacker solicits such codes from the victim, positioned as part of registration or resource download from site-A. If site-B sends CAPTCHA challenges (Section 3.8) to counter automated attacks, they are similarly relayed to the victim on site-A. The attack requires synchronization and seems complicated, but was demonstrated on a number of major real-world services. The attack fails for sites sending reset messages to email addresses on record, but resets relying on secret questions or SMS codes are common, e.g., to address users having lost access to recovery email accounts, or when the account host is itself an email provider and users lack alternate email.

3.4 One-time password generators and hardware tokens

A major security issue with ordinary passwords is their static nature. If observed and captured by a passive attacker (eavesdropper), simple replay of the password defeats security. A step forward is *one-time passwords* (OTPs)—passwords valid for one use only. A challenge is how to pre-share lists of one-time passwords between the party to be authenticated (prover) and the verifier. For electronic account access, some banks give customers paper lists of passwords to be used once each (then crossed off); the server keeps corresponding records for verification. Another method is to use one-way hash functions to generate sequences of one-time passwords from a seed (Lamport, below).

OTPs RECEIVED BY PHONES. Phones (above) can be used as an independent channel for receiving one-time codes. These OTPs are generated system-side and sent to the user's number on record. Beyond use for account recovery, they can also serve as a (standalone) password alternative, or to augment passwords (next section).

OTPs FROM LAMPORT HASH CHAINS. For user A to authenticate to server B, a sequence of one-time passwords can be generated beginning for a random seed secret w as follows (Chapter 2). Using a one-way hash function H, and $t = 100$, consider w, $H(w)$, $H(H(w))$, ..., $H^{t-1}(w)$. For authentication, these will be used once each, in reverse order

$$h_1 = H^{99}(w), h_2 = H^{98}(w), \ ..., \ h_{98} = H(H(w)), \ h_{99} = H(w), \ h_{100} = w \qquad (3.4)$$

So for $1 \leq i \leq 100$ the password for session i is $h_i = H^{t-i}(w)$. As set-up, A sends B $h_0 = H^t(w)$, over a channel ensuring data origin authenticity (so B is sure it is from A, unaltered). Both parties set $i = 1$. Then to authenticate in session i, A computes and sends (id_A, i, h_i). B takes the received value h_i, hashes it once to $H(h_i)$, checks equality to h_{i-1} previously stored, and checks that i matches the expected count. Then i is incremented and h_i is saved for the next verification. After t sessions, the set-up is renewed with a distinct w—as one-time passwords must not be re-used.

‡**Example** *(Pre-play attack on OTPs).* OTP schemes can be attacked by capturing one-time passwords and using them before the system receives the value from the legitimate user. Such attacks have been reported—e.g., an attacker socially engineers a user into revealing its next 5 banking passwords from their one-time password sheet, "to help with system testing". Thus even with OTPs, care should be exercised; ideally OTPs would be sent only to trusted (authenticated) parties.

‡**Example** *(Alternative OTP scheme).* The following might be called a "poor man's" OTP scheme. Using a pre-shared secret P, A sends to B: $(r, \ H(r,P))$. B verifies this using its local copy of P. Since a replay attack is possible if r is re-used, r should be a time-varying parameter (TVP) such as a constantly increasing sequence number or time counter, or random number from a suitably large space with negligible probability of duplication. As a drawback, a cleartext copy of the long-term secret P is needed at B.

‡**Exercise** (Forward guessing attack). Explain how the method of the example above can be attacked if P is a "poorly chosen" secret, i.e., can be guessed within a feasible number of guesses (here "feasible" means a number the attacker is capable of executing).

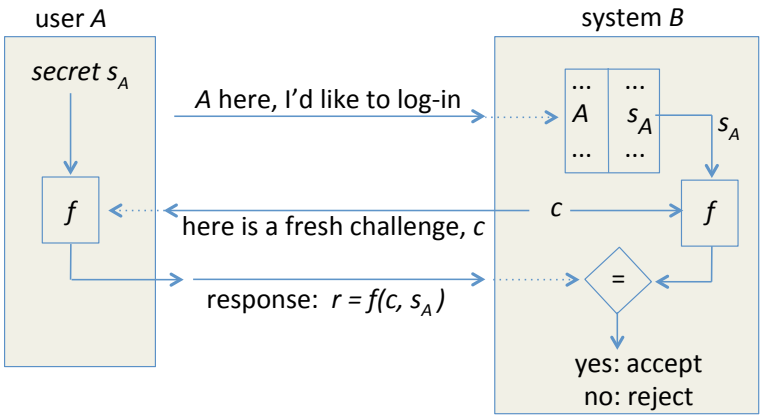

Figure 3.2: Passcode generator using a keyed one-way function f. User-specific secret s_A is shared with the system. Response r is like a one-time password.

PASSCODE GENERATORS. A commercial form of one-time passwords involves inexpensive, calculator-like *passcode generators* (Fig.3.2). These were originally specialized devices, but similar functionality is now available using smartphone apps. The device holds a user-specific secret, and computes a *passcode* output with properties similar to OTPs. The passcode is a function of this secret and a TVP challenge. The TVP might be an explicit (say 8-digit) string sent by system to user for entry into the device (in this case the device requires a keypad). Alternatively, the TVP can be an implicit challenge, such as a time value with, say, one-minute resolution, so that the output value would remain constant for one-minute windows; this requires a (loosely) synchronized clock. The OTP is typically used as a "second factor" (below) alongside a static password. The user-specific secret is stored in cleartext-recoverable form system-side, to allow the system to compute a verification value for comparison, from its local copy of the secret and TVP. Generating OTPs locally via passcode generators, using a synchronized clock as an implicit challenge, can replace system-generated OTPs transmitted as SMS codes to users.

HARDWARE TOKENS. Passcode-generators and authentication means involving mobile phones are specific instances of "what-you-have" alternatives to passwords. A subclass of "what-you-have" authentication methods involves *hardware tokens*, including USB keys, chip-cards (smart cards) and other physical objects intended to securely store and convey secrets—or generate digital tokens (strings) from these secrets—used for authentication. The term *authenticator* is a generic descriptor for a hardware- or software-based means producing secret-based strings used for authentication. The secrets, e.g., symmetric or asymmetric (private) keys, are often used in cryptographic protocols.[10]

MULTIPLE FACTORS. This chapter discusses several alternatives to passwords. Such methods can also be used alongside passwords to *augment* rather than replace them. In

[10] Authentication protocols are discussed in Chapter 4.

simplest form, two methods used in parallel both must succeed for authentication success. In more complex *threshold schemes*, e.g., 3 out of n means must succeed; alternatively, each means contributes to an overall confidence score with success requiring some minimum total. Such additional means are called *second factors* when they require some form of explicit user involvement or action. *Two-factor authentication* involves use of two such means; *multi-factor authentication* is self-explanatory. Ideally, the factors are independent in the sense that if an attack successfully defeats one—such as intercepting a one-time code sent over SMS—a different attack would be needed to defeat the second. If authentication is user-to-device then device-to-web, it is called *two-stage authentication*.

‡**SELECTING FACTORS.** Static passwords are typically the first factor, offering an inexpensive default choice from the "what you know" category. Further factors are often from distinct traditional categories—what you have (hardware tokens, passcode generators, other devices) and what you are (biometrics, below). Thus common two-factor schemes are (password, biometric) or (password, OTP from passcode-generator). Automated banking machines commonly require (PIN, chip-card)—something you know plus something you have. If, against advice, you write your PIN on the card itself, the two factors are no longer independent and a single theft allows defeat.

‡**COMPLEMENTARY FACTORS AND PROPERTIES.** Multiple factors should be combined with an eye to the complementary nature of the resulting combined properties. Using two what-you-know factors (two passwords) increases memory burden; a hardware token avoids the cognitive burden of a second password. However, hardware authenticators must be carried—imagine having as many as passwords! When multiple schemes are used in parallel, if they are independent (above), their combined security is at least that of the weaker, and ideally stronger than each individually—otherwise there is little benefit to combine them. Regarding usability however, inconveniences of individual factors are typically also additive, and the same is true for deployability barriers/costs.

‡**SIGNALS VS. FACTORS.** Some system use "invisible" or "silent" authentication checks behind-the-scenes, which do not require explicit user involvement. Beyond earlier-discussed *authentication factors*, which require explicit user actions, the broader class of *authentication signals* includes also implicit means such as:

- IP-address-checks of devices previously associated with successful logins;
- browser cookies stored on devices after previously successful authentication;
- *device fingerprinting*, i.e., means to identify devices associated with legitimate users (previous successful logins), by recognizing hardware or software characteristics.

Individual signals may involve secrets assumed known or possessed only by legitimate users, or devices or locations previously associated with legitimate users. Silent signals offer usability advantages, as long as they do not trigger false rejection of legitimate users.

‡**FACTORS, PRIMARY AUTHENTICATION, AND RECOVERY.** Are second factors suitable for stand-alone authentication? That depends on their security properties—but for a fixed application, if yes, then there would appear to be little reason to use such a

factor in combination with others, except within a thresholding or scoring system. As a related point, any password alternative may be suitable for account recovery if it offers sufficient security—but in reverse, recovery means are often less convenient/efficient (which is tolerable for infrequent use) and therefore often unsuitable for primary authentication. Secret questions are thus used mainly for account recovery; and as discussed, their security makes even that questionable when used stand-alone.

3.5 Biometric authentication

Certain human characteristics are unique to individuals, even across large populations. *Biometric authentication* methods leverage this. *Physical biometrics* (based on static physiological characteristics) provide the "what you are" category of authentication, while *behavioural biometrics* (based on behavioural features related to physiology) provide part of a "what you do" category. Behavioural characteristics independent of human physiology, such as geolocation patterns and call-patterns (phone numbers called), can also be used in non-biometric *behavioural authentication* approaches. A set of biometric features that can be used for authentication is called a biometric *modality*.

BIOMETRIC MODALITIES. Example modalities of physical biometrics are:

- fingerprints,
- facial recognition,
- iris recognition (i.e., involving part of the eye that a contact lens covers),
- hand geometry (e.g., based on length, size, and shape of fingers and palm), and
- retinal scan (i.e., based on patterns of retinal blood vessels).

As examples, consumer devices such as laptops and mobile phones have used fingerprint and face biometrics; both fingerprint (four digits) and iris-based biometrics have been used at U.S.-Canada airport borders. Example modalities of behavioural biometrics are:

- voice print (a physical-behavioural mix),
- typing rhythm (i.e., keystroke patterns and timing),
- mouse and scrolling patterns, or swipe patterns (on touch-screen devices); and
- gait (characteristics related to walking).

BIOMETRICS ARE NON-SECRETS. Authentication approaches that rely on demonstrating knowledge of a secret, such as passwords, rely on the assumption that the secret is known only to authorized parties. (That this assumption is commonly violated for passwords is a security weakness.) Biometric characteristics are not secrets—fingerprints are left on many surfaces, and faces are readily visible. Thus biometric authentication relies

on a different assumption: that samples are input by a means providing some assurance of being "tightly bound" to the user present. Consider a fingerprint sampled at a supervised entrance, e.g., at an airport or corporate facility. The supervision provides some assurance that the individual is not presenting an appendage other than their own for physical measurement, nor injecting a data string captured earlier. Biometrics thus implicitly rely on some form of *trusted input channel*; this tends to make them unsuitable for remote authentication over the Internet. Note that your iPhone fingerprint is not used directly for authentication to remote payment sites; a two-stage process involves user-to-phone authentication (biometric verification by the phone), then phone-to-site authentication (using a protocol leveraging cryptographic keys).

ADVANTAGES (BIOMETRICS). While pros and cons of biometrics vary depending on the modality, some general statements can be made. Biometrics have usability advantages over alternatives: nothing need be carried or remembered, they are generally easy to use, and they offer scalability in terms of reduced burden on users (unlike passwords, the burden does not increase with the number of accounts). These are powerful advantages.

DISADVANTAGES (BIOMETRICS). Many modalities require custom client-side hardware; to accommodate rejection of legitimate users, failure-to-enrol and failure-to-acquire (below), fall-back mechanisms are typically needed, further adding to system complexity and cost. Scalability has a related downside: if fingerprints are used across many systems, each taking samples and storing templates, a scenario analogous to re-using passwords results: a compromise from any one system can put the others at risk. From experience, believing no such compromise will occur is naive; but here, because a foundational requirement for biometrics is that they cannot be changed, the consequences are severe. This inherent "feature" of biometrics being unrevokable is a daunting show-stopper. Moreover, as biometrics are non-secrets, their "theft" does not require breaking into digital systems—thus the criticality of ensuring fresh samples bound to individuals, and trusted input channels. Aside from this, the security of biometrics is often over-stated—unique biometric characteristics across individuals does not preclude system circumvention; security often depends more on system implementation details than modality.

Example *(iPhone fall-back authentication).* Biometric authentication is generally considered more secure than short numeric passwords (PINs). In 2013, iPhone fingerprint authentication replaced (typically 4-digit) login PINs. Face recognition on 2017 iPhones replaced this using 3D face models. If such systems are built to fall-back to ordinary PIN authentication if the biometric fails to recognize the user after several tries, then the overall system is no stronger than the fall-back mechanism.

SUMMARY. Biometrics offer usability advantages, have some deployability drawbacks, are often less secure than believed, and have failure modes with severe negative externalities (i.e., consequences on unrelated parties or systems). Thus, biometrics are by no means a "silver bullet" solution. Their suitability depends, as usual, on the target environment of use; they suit supervised environments better than remote authentication.

BIOMETRIC PROCESS: ENROLMENT AND VERIFICATION. A biometric modality is implemented by selecting a suitable set of measurable *features*—e.g., for fingerprints, the arches, loops and whorl patterns formed by skin ridges, their length, relative locations

and distances between them. For each user (account), several sample biometric measurements are taken in an enrolment phase. Features are extracted to build a *reference template*. For subsequent user authentication, a freshly taken sample is compared to the template for the corresponding implied or asserted account, and a *matching score* s is computed; higher scores indicate higher similarity, e.g., $s = 0$ could denote no similarity, with $s = 100$ denoting 100% agreement. A threshold t is set (discussed later). Then if $s \geq t$, the system declares the sample to be from the same individual as the template.

Exercise (Biometric system flow chart). Illustrate the process of biometric enrolment and verification in a flow-chart relating architectural components. (Hint: see [33, Fig.1])

FALSE REJECTS, FALSE ACCEPTS. Two types of errors occur in biometric systems. In a *false reject*, a legitimate user's new sample is declared to not match their own template. In a *false accept*, an imposter's sample is (wrongly) declared to match the legitimate user's template. The frequency of these errors depends on both the threshold t and system limitations (inaccuracies in sampling, measurement and feature representation). Measurement accuracy is affected by how user features present to sensors; environmental factors also come into play, e.g., dry skin from cold weather impacts fingerprint readings.

A stricter threshold (larger t, requiring stronger matches) results in more false rejects, but fewer false accepts; this negatively affects usability and availability, but improves security. A looser tolerance (smaller t, accepting weaker matches) results in fewer false rejects, but more false accepts; this improves usability and availability for legitimate users, but obviously decreases security. What is acceptable as a tradeoff between false accepts and false rejects depends on the application; t is adjusted to suit application scenarios. High-security (security-sensitive) applications demand stricter matching, tolerating more false rejects in order to preserve security; low-security applications prioritize usability over security, setting looser tolerances in order to reduce false rejects.

FALSE ACCEPT/REJECT RATES. For a fixed threshold t and fixed legitimate user L with reference template X_L, let X_V denote biometric samples to be matched. The *false accept rate* (FAR) is the probability of a false accept, i.e., prob(system declares X_V matches X_L, when in fact X_V is not from L); the probability assumes sampling over the user population. Theoretically, to determine a system FAR, the above could be computed over all users L and reported in composite.

Aside: we expect serious attackers will do better than using random samples in impersonation attempts, but the FAR statistic reflects such a random sampling. This may be viewed as reflecting a naive attacker, or benign errors. Security researchers thus view the FAR as misleadingly optimistic, giving more weight to resilience under malicious scenarios (see "circumvention", below).

The *false reject rate* (FRR) is the probability of a false reject, i.e., prob(system declares X_V does not match X_L, when sample X_V is actually from L); sampling is over repeated trials from user L. The *equal error rate* (EER) is the point at which FAR = FRR. Although unlikely to be the preferred operational point in practice, EER is used for simplified single-point comparisons—the system with lower EER is preferred.

TWO-DISTRIBUTION OVERLAP: USER/INTRUDER MATCH SCORES. To illustrate how altering the threshold t trades off false accepts and false rejects, we consider two

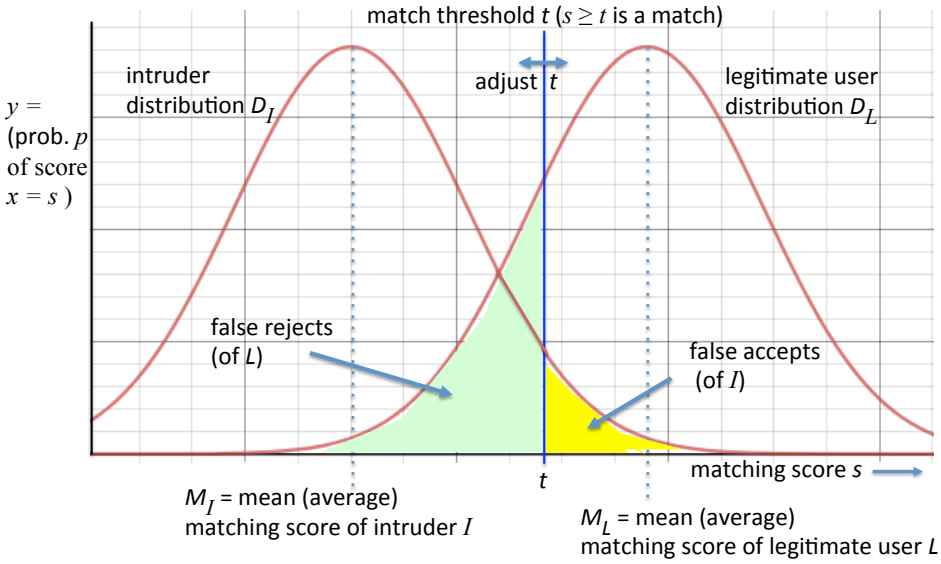

Figure 3.3: Biometric system trade-offs: false accepts vs. false rejects. The curves are illustrative probability distributions for an intruder and legitimate user's matching scores; higher scores indicate better match to the user's biometric template. The y axis reflects the relative number of biometric sample events that would receive matching score $x = s$.

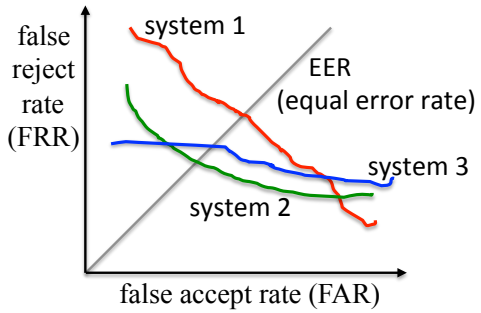

Figure 3.4: DET graph (Detection Error Tradeoff). A DET graph shows a system's *operating characteristics* across implicit values for the matching threshold t. If the EER is used as a single point of comparision, then System 2 is preferred. System 3's FRR decreases slowly as system parameters are adjusted to admit a higher FAR; in contrast, System 1's FRR decreases more rapidly in return for an increased FAR.

graphs for a fixed user L with template X_L. The first (Fig.3.3) has an x-axis giving matching score of samples (against X_L), and y-axis for probability of such scores across a large set of samples. Two collections of samples are shown, giving two probability distributions: D_I (on left) for intruder samples, D_L (on right) for samples from user L. Each value of t defines two shaded areas as shown, respectively corresponding to false rejects and false accepts; moving t left or right changes the relative sizes of the shaded areas (trading off false rejects and false accepts). Noting each shaded area as a percentage of its corresponding distribution and interpreting this as a rate allows drawing a second graph (Fig.3.4). Its x-axis denotes FAR, y-axis FRR, and curve point (x,y) indicates FRR = y when FAR = x. Each point implicitly corresponds to a value t in the system of the first graph. The second, called a *detection error tradeoff* (DET) graph, thus shows the system's possible *operating characteristics* across implicit values t. Such analysis is common in binary classifier systems.[11]

FAILURE TO ENROL/CAPTURE. *Failure to enrol* (FTE) refers to how often users are unsuccessful in registering a template. For example, a non-negligible fraction of people have fingerprints that commercial devices have trouble reading. The FTE-rate is a percentage of users, or percentage of enrolment attempts. *Failure to capture* (FTC) refers to how often a system is unable to acquire a sample of adequate quality to proceed. FTE-rate and FTC-rate should be examined jointly with FAR/FRR rates, due to dependencies.

EVALUATING BIOMETRICS USING STANDARD CRITERIA. To evaluate and compare biometric systems, a suite of properties and standardized criteria are used. Basic requirements must first be met for a modality to be given serious consideration for biometric authentication. The following aspects of a modality's characteristics are considered.

- *universality*: do all users have the characteristic? This relates to failure-to-enrol.
- *distinguishability*: do the characteristics differ sufficiently across pairs of users to make benign matches unlikely? This requires sufficient variability in measurable features, and impacts false accept rates.
- *invariance*: are characteristics stable over time (even for behavioural biometrics)?
- *ease-of-sampling*: how easily are samples obtained and measured? For example, consider DNA vs. fingerprints. Retinal scans typically involve contact with an eyepiece. Physical biometrics may be obscured (e.g., by hair, glasses).

Beyond basic requirements, other criteria important for use in practice are as follows.

- *accuracy*: metrics discussed earlier include FAR, FRR, EER, FTE-rate, FTC-rate (all for selected thresholds t). These reflect operation in anticipated or target operating environments and conditions, albeit with benign participants.

[11] DET graphs are closely related to ROC curves. Both arise in analyzing four diagnostic outcomes of a binary classifier (true/false positive, true/false negative); false positive and false negative are the error outcomes. ROC is short for *relative operating characteristic* or (less intuitively) *receiver operating characteristic*.

- *cost*: this includes time (sampling; processing), storage, hardware/software costs.

- *user acceptance*: are users willing to use the system (positively affected)? Some users have concerns about privacy of biometric data in general, or its potential use for tracking. Examples of modality-specific concerns are invasiveness (e.g., fear or concern about light or objects near the eyes), and negative cultural associations (e.g., some cultures associate fingerprinting with implied criminal activity).

- *attack-resistance*: can the system avoid adversarial false accepts, i.e., resist user impersonation (spoofing), substitution, injection, or other attempted circumvention?

CIRCUMVENTION: ATTACKS ON BIOMETRIC AUTHENTICATION. The basic security question for a biometric system is: how easily can it be fooled into accepting an imposter? This asks about malicious false accepts, whereas FAR measures benign false accepts. Related questions are: What attacks work best, are easiest to mount, or most likely to succeed? How many authentication trials are needed by a skilled attacker? These questions are harder to address than those about performance measures noted above.

Exercise (Circumventing biometrics). (i) Outline generic system-level approaches to defeating a biometric system, independent of the modality. (Hint: see [32, Fig.9].) (ii) For each of five selected modalities from those listed earlier, summarize known modality-specific attacks. (iii) For which modalities are liveness detectors important, or possible?

BIOMETRICS: AUTHENTICATION VS. IDENTIFICATION. This section has considered biometrics mainly for *user authentication*, e.g., to replace passwords or augment them as a second factor. In this usage, a user (account) is first asserted, then the biometric sample is matched against the (single) corresponding user template. An alternate usage is *user identification* (i.e., without asserting specific identity); the system then must do a one-to-many test—as explained in this chapter's first paragraph. For local identification to a laptop or smartphone, the number of accounts registered on that device is typically small; access is granted if any match is found across registered accounts, and the one-to-many matching has relatively negligible impact on computation (time) or security. Relieving the user of the task of entering a username is done to improve convenience.

However for systems with large user bases, one-to-many matching is a poor fit with access control applications. The probability of benign match between a single attacker-entered sample and *any one of the many* legitimate user templates is too high. The natural application of "user identification" mode is, unsurprisingly, identification—e.g, to match a target fingerprint against a criminal database, or video surveillance face recognition to match crowd faces against a targeted list (the shorter the better, since the latter case is many-to-many). The issue of false accepts is handled here by using biometric identification as the first-stage filter, with human processing for second-stage confirmation.

Exercise (Comparing modalities). Select six biometric modalities from the bulleted list earlier in this section, plus two others not listed. (i) For each, identify primary advantages and limitations. (ii) Using these modalities as row labels, and bulleted criteria above as column labels, carry out a qualitative comparison in a table, assigning one of

(low, medium, high) to each cell; word the criteria uniformly, such that "high" is the best rating. (iii) For each cell rated "low", briefly justify in one-sentence. (Hint: see [33], [8].)

3.6 ‡Password managers

Password managers are software tools that store and retrieve passwords. Some auto-fill web site userid-password pairs. Instead of remembering a large number of passwords, a user remembers one *master password*, which provides access to the others.[12] Efficiency and usability improve, with reduced memory burden. Resilience to phishing improves if the tool records the domain associated with outgoing passwords, and disallows (or warns) on subsequent attempts to send a password to a domain not previously associated with it. Ideally, security improves by allowing users to choose a master password with high guessing-resistance, and then random (unguessable) individual site passwords which need no longer be remembered manually. In practice, master passwords may be weaker than hoped, and the individual site passwords managed remain not only static (thus replayable) but often remain user-chosen (thus guessable) for reasons explained below. Overall, password managers thus deliver fewer security advantages than expected.

SOFTWARE PACKAGING. A password manager may be integrated with the operating system (macOS Keychain uses the OS login password as master password), or be a standalone client application, a browser built-in feature or plug-in/add-on, or a cloud-based service. Some managers *synchronize* passwords across devices, i.e., make them accessible from (stored encrypted on) multiple designated user devices; otherwise, the passwords managed are available only on a primary device.

PASSWORD MANAGER APPROACHES. The two main approaches are as follows.

1. *password wallet* (or *vault*): here the tool manages an existing collection of passwords, automatically selecting the password needed based on association with the domain of use. Passwords are stored in the wallet individually encrypted under a password derived from the master password; very weak tools leave them plaintext. The master password is entered at the start of each manager session (e.g., start of day); caching it allows later use without re-entry by the user. Since the tool now remembers site passwords rather than the user, and can generate long, random passwords, the main barriers to making all site passwords unguessable should be gone; but in practice, wallets typically manage pre-existing passwords, because "migrating" existing passwords site-by-site to new random passwords consumes user time. Thus many password wallets manage existing (guessable) passwords. The wallet is stored on the local device in client-side tools, or at a server in cloud-based tools.

2. *derived passwords*: here application-specific or site-specific passwords are derived from a master password plus other information such as target domain. This provides some protection against *phishing attacks*; a rogue site attracting a target user

[12] Alternate strategies for coping with overwhelming numbers of passwords include regularly relying on password reset mechanisms (above), and use of single sign-on architectures (Chapter 4).

receives the string H(master, rogue.domain.com) rather than the authorizing string H(master, true.domain.com) that allows account access on the true domain. Derived site passwords (vs. user-chosen) can increase guessing-resistance through (client-side) iterated hashing; and protect against pre-computed dictionary attacks by using user-specific salts (Section 3.2).

Exercise (Offline attack on master password). The phishing protection just mentioned (under derived passwords) comes with a risk: a rogue site may mount an offline guessing attack to recover a user master password. Explain how this is done, and what factors affect the likelihood of attacker success.

SECURITY AND RISKS. Password managers are password "concentrators", thus also concentrating risk by creating a single point of failure and attractive target. Threats to the master password include capture (e.g., by client-side malware, phishing, and network interception), offline guessing (user-chosen master passwords), and online guessing in the case of cloud-based services. Individual site passwords managed, unless migrated (below) to random passwords, remain subject to guessing attacks.

RISK IF PASSWORD MANAGER FAILS. Once a password generator is used to generate or remember passwords, users rely on it (rather than memory); if the tool becomes unavailable or malfunctions, any password recovery mechanisms in place through a web site may allow recovery access of (some of) the managed passwords. However, typically no such recovery service is available for the master password itself, nor any managed passwords used for password-derived keys for stand-alone applications, e.g., local file/disk encryption. If access to such a password is lost, you should expect the locally encrypted files to be (catastrophically) unrecoverable.

COMPATIBILITY WITH EXISTING PASSWORD SERVERS. An advantage of password wallets is that for managing existing passwords, they introduce no server incompatibilities and thus can be deployed without any server-side changes or co-operation. In the case of generating new (random) passwords, both password wallet managers and derived-password managers must satisfy server-defined password composition policies—automatically generated passwords will not always satisfy policies on first try. Thus derived-password managers cannot regenerate site passwords on-the-fly from master passwords alone; they may still need to store, for each user, site-specific information beyond standard salting info, as a type of additional salt to satisfy site-rules.

Exercise (Analysis and user study of password managers). For each of (i) PwdHash, and (ii) Password Multiplier, answer the following. (a) Explain the technical design of this manager tool, and which manager approach it uses. (b) Summarize the tool's strengths and weaknesses related to each of: security, usability, deployability. In particular for usability, describe how users invoke the tool to protect individual site passwords, and for any automated actions, how users are signalled that the tool is operating. (c) Describe how the tool performs on these standard password management tasks: day-to-day account login, password update, login from a new device, and migration of existing passwords to those compatible with the manager tool. (Hint: see [13].)

3.7 ‡Graphical passwords

In *graphical password* schemes, passwords depend in some way on pictures or patterns. Based on the idea that human memory is better for pictures than text, the goal is to improve usability, and especially ease memory burden compared to regular passwords (character strings). Ideally security can simultaneously be increased, if the reduced memory burden allows users to choose harder-to-guess passwords. Like regular passwords, the resulting password is encoded into a string that can be verified by the system.

As a simple example, Android touch-screen devices commonly use a swipe pattern over a 9-dot background to replace a login PIN/password. This falls into the category of a *pure recall* scheme—the user is essentially reconstructing a pattern on a blank sheet.

A second category is *cued-recall* schemes, for example presenting a user with a picture and asking them to choose 5 click-points as their password. The user must later re-enter those points (within reasonable tolerance) to gain account access.

A third category is *recognition* schemes, for example presenting a user with 4 panels sequentially, each with 9 faces: 8 *distractors* and one face familiar to the user (a set of familiars is selected during registration). The user must click on a familiar face in each of the 4 panels; other sets of common objects can be used instead of faces, e.g., house-fronts. From cognitive psychology research, people are better at recognizing previously encountered items (*recognition memory*) than in tasks involving (pure) recall.

Analysis of graphical password schemes to date has found the following.

1. The best among proposals offer minor usability (memorability) improvements. In some cases this is due to strategies that could similarly be used with text passwords (e.g., cued-recall). Many proposals require significant training, longer password registration phases, and moderately longer password entry times.

2. Promised security improvements are typically elusive due to the tendency of users to choose predictable graphical passwords—some are typically much more popular than others—just as users choose predictable text passwords. (The ideal is equi-probable passwords from a suitably large password space—e.g., for text passwords, consisting of characters chosen truly at random; for cryptographic keys, the ideal is truly random secret bit strings.)

3. Overall, even where there may be minor advantages over text passwords, these appear to be insufficient to displace text passwords on a broad scale.

From a security viewpoint, in the end many graphical password schemes boil down to fixed authentication strings, which if captured, are replayable like captured text passwords. More compelling benefits are likely needed for an alternative to displace text passwords.

3.8 ‡CAPTCHAs (humans-in-the-loop) vs. automated attacks

Some web sites ask users to type in text corresponding to distorted character strings. This is an example of a CAPTCHA,[13] also called an *Automated Turing Test* (ATT). The idea is to present a task relatively easily done by humans, but difficult for computer programs. Such tests are used in security to distinguish humans from malicious programs ("robots" or *bots*) which might try to acquire in bulk free online resources such as e-mail accounts (e.g., to send spam email), or make bulk postings of spam or malware to online discussion boards. CAPTCHAs are commonly based on character recognition (CR), audio recognition (AUD), image recognition (IR), or cognitive challenges involving puzzles/games (COG). Below we explore (Figure 3.5) how they can help stop automated online-guessing attacks.

As noted earlier, to mitigate online guessing attacks, a server may limit (*throttle*) the number of online login attempts. However, if account "lock-out" results, this inconveniences the legitimate users of attacked accounts. A specific attacker goal might even be, e.g., to lock out a user they are competing with precisely at the deadline of an online auction. A defensive alternative is to make each login guess "more expensive", by requiring a correct ATT response accompany each submitted password—but this inconveniences legitimate users. The cleverly designed protocol outlined next does better.

PINKAS-SANDER LOGIN PROTOCOL. The protocol imposes an ATT on only a fraction p of login attempts (and always when the correct password is entered but the device used is unrecognized). It assumes legitimate users typically log in from a small set of devices recognizable by the server (e.g., by setting browser cookies or *device fingerprinting*); and that any online dictionary attack is mounted from other devices. Device recognition is initialized once a user logs in successfully. Thereafter the legitimate user faces an ATT only when either logging in from a new device, or on a fraction p of occurrences upon entering an incorrect password.

In Figure 3.5, note that requiring an ATT only upon entry of the correct password would directly signal correct guesses. Also, whether to ask an ATT for a given password candidate must be a deterministic function of the submitted data, otherwise an attack program could quit any login attempt triggering an ATT, and retry the same userid-password pair to see if an ATT is again asked or if a "login fails" indication results.

Since it is expected that an attacker will make many incorrect guesses, imposing an ATT on even a small fraction of these (e.g., 5%) is still a large cost. The attacker, assumed to be submitting guesses from an unrecognized machine, must always "pay" with an ATT on submitting a correct guess, and must similarly pay a fraction p of the time for incorrect guesses. But since the information available does not reveal (before answering the ATT) whether the guess is correct, abandoning an ATT risks abandoning a correct guess.

Exercise (Pinkas-Sander password protocol analysis). This protocol (Figure 3.5) can be analyzed under two attack models: (i) an automated program switches over to a human attacker to answer ATTs; (ii) the program makes random guesses as ATT answers, assum-

[13] The acronym originated from *Completely Automated Public Turing test to tell Computers and Humans Apart*, but as common implementations are often proprietary, the middle part may better be *Program to Tell*.

```
1  fix a value for system parameter p, 0 < p ≤ 1 (e.g., p = 0.05 or 0.10)
2  user enters userid-password pair
3  if (user PC has cookie) then server retrieves it
4  if (entered userid-password correct) then
5     if (cookie present & validates & unexpired & matches userid) then
6        "login passes"
7     else % i.e., cookie failure
8        ask an ATT; "login passes" if answer correct (otherwise "login fails")
9     endif
10 else % i.e., incorrect userid-password pair
11    set AskAnATT to TRUE with probability p (otherwise FALSE) †
12    if (AskAnATT) then
13       ask an ATT; wait for answer; then independent of answer, say "login fails"
14    else immediately say "login fails" endif
15 endif
```

Figure 3.5: Protocol to counter online dictionary attacks (simplified Pinkas-Sander [46]). †Setting is a deterministic function of userid-password pair (same each time for that pair)

ing an ATT answer space of n elements (so an ATT guess is correct with probability 1 in n). To simplify analysis, assume a space of S equi-probable passwords. (a) Under model (i), for an optimal attacker, determine the expected number of ATTs answered before successfully guessing a password; express your answer as a function of p and S, and assume an attack on a single account. (b) Under model (ii), determine the expected number of password guesses needed before success, as a function of p, S and T. (Hint: see [46].)

CAPTCHA FUTURES. For several reasons, the ongoing value of CAPTCHAs in security is unclear. For many types of CAPTCHAs, automated solvers are now so good that CAPTCHA instances sufficiently difficult to resist them are beyond the annoyance and complexity level acceptable for legitimate users—so these CAPTCHA cease to be useful Turing Tests. The efficacy of CR CAPTCHA solvers in particular has resulted in more IR CAPTCHAs. Another attack on CAPTCHAs is maliciously outsourcing them by redirection to unsuspecting users. Similarly, the core idea of distinguishing humans from bots is defeated by redirecting CAPTCHAs to willing human labour pools—"sweat shops" of cheap human solvers, and Amazon Mechanical Turkers.

Example *(Google re*CAPTCHA*).* In 2014, the Google reCAPTCHA project replaced CAPTCHAs with checkboxes for users to click on, labelled "I'm not a robot". A human-or-bot decision is then made from analysis of browser-measurable elements (e.g., keyboard and mouse actions, click locations, scrolling, inter-event timings). If such first-level checks are inconclusive, a CR or IR CAPTCHA is then sent. In 2017 even such checkboxes were removed; apparently the actions triggered by requesting clicking of a checkbox have been superceded by pre-existing measurable actions or other recognition means.

Information	Probability	Hex representation	Binary alternative
red	0.25	0000	00
green	0.25	00FF	01
blue	0.25	FF00	10
black	0.25	FFFF	11

Table 3.1: Alternative representations for conveying four known values. The same information is conveyed, whether two bytes of data are used to represent it, or two bits.

3.9 ‡Entropy, passwords, and partial guessing metrics

This section discusses information-theoretic (Shannon) entropy, guessing entropy and partial guessing metrics useful for understanding guessing attacks on user-chosen passwords.

Example *(Data, information representation, and entropy).* A 16-bit word might be used to convey four values (Figure 3.1), representing events that are equi-probable over a large number of samples. The same information can be conveyed in 2 bits. For the given probabilities, in information theory we say there are two bits of *entropy*. Below, we explain further, using password distributions both for concreteness and relevance.

SHANNON ENTROPY. Let $q_i > 0$ be the probability of event x_i from an event space $\mathcal{X}$ of n possible events ($1 \leq i \leq n$, and $\sum q_i = 1$). In our exposition, $x_i = P_i$ will be a user-chosen password from a space of n allowable passwords, with the set of passwords chosen by a system's m users considered an experimental outcome. In math-speak, a *random variable* X takes value $x_i = P_i$ with probability q_i, according to a *probability distribution* D_X. We write $X \xleftarrow{D_X} \mathcal{X}$ and D_X: $q_i \rightarrow x_i$. D_X models the probability of users choosing specific passwords, e.g., as might be derived from an unimaginably large number of real-world iterations. Now the *Shannon entropy* of this discrete distribution is defined as:[14]

$$H(X) = H(q_1, q_2, ..., q_n) = \sum_{i=1}^{n} q_i \cdot lg(1/q_i) = -\sum_{i=1}^{n} q_i \cdot lg(q_i) \tag{3.5}$$

where the units are *bits of entropy*, lg denotes a base-2 logarithm, and by convention $0 \cdot lg(0) = 0$ to address $lg(0)$ being undefined.[15]

$H(X)$ measures the *average uncertainty* of X. $H(X)$ turns out to be the minimum number of bits needed (on average, across the probability distribution) to convey values $X = x_i$, and the average wordlength of a minimum-wordlength code for values of X.

INTERPRETATION OF ENTROPY. To help understand the definition of $H(X)$, for each outcome x_i define $I(x_i) = -lg(q_i)$ as the amount of *information* conveyed by the event $\{X = x_i\}$. It follows that the less probable an outcome, the more information its observation conveys; observing a rare event conveys more than a common event, and observing an event of probability 1 conveys no information. The average (expected value)

[14] Note that only the probabilities q_i are important, not the events themselves.

[15] Here in Section 3.9, by tradition, H denotes the Shannon entropy function (not a hash function).

of the random variable I is then $H(X) = E_X(I_X) = E_X(-lg(q_i))$. Viewing q_i as a weight on $lg(q_i)$, $H(X)$ is now seen to be the expected value of the log of the probabilities.

ENTROPY PROPERTIES. The following hold for $H(X)$ with event space of size n.

1. $H(X) \geq 0$. The minimum 0 occurs only when there is no uncertainty at all in the outcome, i.e., when $q_i = 1$ for some i (forcing all other q_j to 0).

2. $H(X) \leq lg(n)$. The maximum occurs only when all $q_i = \frac{1}{n}$ (all events equi-probable). Then $H(X) = \sum_{i=1}^{n} 1/n \cdot lg(n) = lg(n)$. Thus a *uniform* ("flat") distribution maximizes entropy (gives greatest uncertainty), e.g., randomly-chosen cryptographic keys (whereas user-chosen passwords have highly skewed distributions).

3. Changes towards equalizing the q_i increase $H(X)$. For $q_1 < q_2$, if we increase q_1 and decrease q_2 by an equal amount (diminishing their difference), $H(X)$ rises.

Example *(Entropy, rolling a die).* Let X be a random variable taking values from rolling a fair 8-sided die. Outcomes $\mathcal{X} = \{1,2,3,4,5,6,7,8\}$ all have $q_i = \frac{1}{8}$ and $H(X) = lg(8) = 3$ bits. For a fair 6-sided die, $q_i = \frac{1}{6}$ and $H(X) = lg(6) = 2.58$ bits. If the 6-sided die instead has outcomes $\mathcal{X} = \{1,2,3,4,5,6\}$ with respective probabilities $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32}$, then $H\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32}\} = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{16} \cdot 4 + 2(\frac{1}{32} \cdot 5) = 1.9375$ bits, which as expected, is less than for the fair die with equi-probable outcomes.

Exercise (Entropy, rolling two dice). Let X be a random variable taking values as the sum on rolling two fair 6-sided dice. Find the entropy of X. (Answer: 3.27 bits.)

Example *(binary entropy function).* Consider a universe of $n = 2$ events and corresponding entropy function $H = -(p \cdot lg(p) + q \cdot lg(q))$, where $q = 1 - p$. A 2D graph (see Fig.3.6) with p along x-axis [0.0, 1.0] and H in bits along y axis [0.0, 1.0] clearly illustrates that $H = 0$ if and only if one event has probability 1, and that H is maximum when $q_i = \frac{1}{n}$. (Compare to: properties above. Source: [50] or [37, Fig.1.1].)

SINGLE MOST-PROBABLE EVENT. The question of single passwords with highest probability has been studied. If an attacker is allowed exactly one guess, the optimal strategy is to guess the most-probable password based on available statistics. A company might analyze its password database to determine the percentage of users using the most popular password, to measure maximum expected vulnerability to a single-guess attack by an "optimal attacker" assumed to know the probability distribution. The expected probability of success is $q_1 = max_i(q_i)$; this assumes the target account is randomly selected among system accounts. A formal measure of this probability of the most likely single event is given by the *min-entropy* formula:

$$H_\infty(X) = lg(1/q_1) = -lg(q_1). \tag{3.6}$$

If $q_i = \frac{1}{n}$ for all i, then $H_\infty(X) = -lg(\frac{1}{n}) = lg(n)$, matching Shannon entropy in this case.

Example *(Most-probable password).* Suppose the most popular password in a distribution is "Password1", with 1% of users choosing it. Knowing this, but not which account(s) use this password, an attacker tries it on a randomly chosen account from this

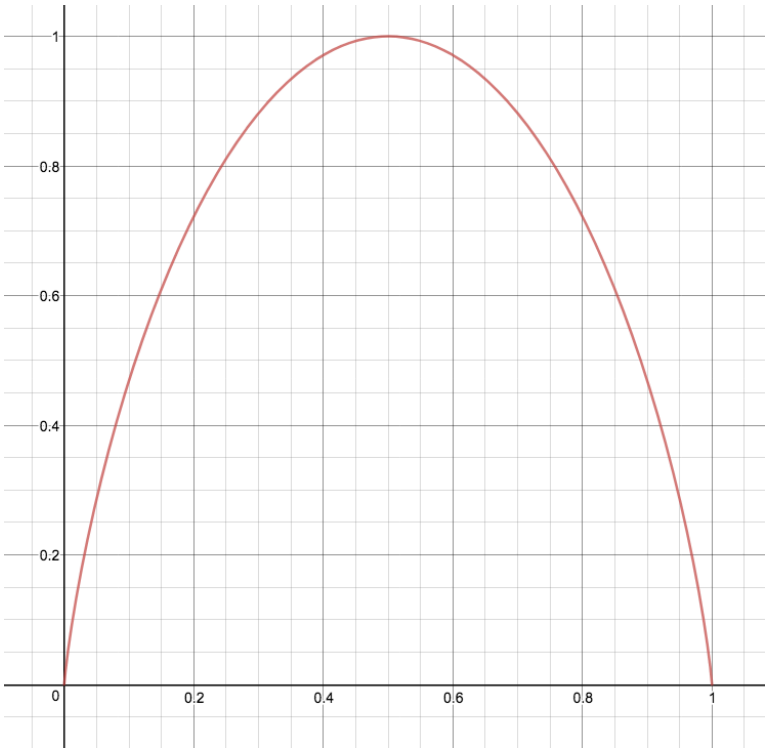

Figure 3.6: Binary entropy function for $n = 2$ events. The x-axis is the probability p of one of the events; $y = H(x) = -x \cdot \log_2(x) + (1-x)\log_2(1-x))$, in bits.

distribution; assume account names are easily obtained. The attacker then has a 1% probability of success with this one guess.

Example *(Optimal breadth-first guessing).* If passwords are ordered $P_1, P_2, \ldots$ in highest to lowest probability order, then for an attacker allowed multiple online guesses, the optimal strategy to minimize expected time to success is to guess P_1 on all accounts, then P_2 on all accounts, and so on. If P_1 has probability 1% per the previous example, then success is expected (50% chance) after guessing P_1 on 50 accounts.

GUESSWORK (GUESSING FUNCTION). With notation as above, let $q_i \geq q_{i+1}$, modeling an optimal guessing attack order. The *guessing index* $g(X)$ over a finite domain X assigns a unique index $i \geq 1$, called a *guess number*, to each event $X = x_i$ under this optimal ordering. Then for $X \xleftarrow{R} \mathcal{X}$ (X drawn randomly from the event universe according to distribution D_X above), the average (expected) number of guesses needed to find X by sequentially asking, in optimal order, "Is $X = x_i$?" is given by the *guessing function*

$$G_1(\mathcal{X}) = E[g(X)] = \sum_{i=1}^{n} i \cdot q_i \quad \text{(units = number of guesses)} \tag{3.7}$$

Like $H(X)$, G_1 gives an expectation *averaged over all events* in $\mathcal{X}$. Thus its measure is

relevant for an attack executing a *full search* to find all user passwords in a dataset—but not one, e.g., quitting after finding a few easily-guessed passwords. If $q_i = 1/n$ for all i,

$$G_1(n \text{ equally probable events}) = \sum_{i=1}^{n} i \cdot 1/n = (1/n)\sum_{i=1}^{n} i = (n+1)/2 \qquad (3.8)$$

since $\sum_{i=1}^{n} i = n(n+1)/2$. Thus in the special case that events are equi-probable, success is expected after guessing about half-way through the event space; note this is *not* the case for user-chosen passwords since their distributions are known to be heavily skewed.

Example *(Guesswork skewed by outliers).* As a guesswork example, consider a system with $m = 32$ million $\approx 32 \cdot 2^{20}$ users, whose dataset $\mathcal{R} \subset \mathcal{X}$ of m non-unique user passwords has a subset $\mathcal{S} \subset \mathcal{R}$ of 32 elements that are 128-bit random strings (on average 1 per 2^{20} elements in $\mathcal{R}$) . Let $\mathcal{U}_{2^{128}}$ denote the set of all 128-bit random strings. From (3.8), $G_1(\mathcal{U}_{2^{128}}) > 2^{127}$. Per individual password in $\mathcal{S}$ we thus expect to need at least 2^{127} guesses. How does this affect G_1 overall? From (3.7) and averaging estimates,[16] it follows that $G_1(\mathcal{R}) > 2^{127} \cdot 2^{-20} = 2^{107}$ guesses independent of any passwords outside $\mathcal{S}$. Thus the guesswork component from difficult passwords swamps (obscures) any information that G_1 might convey about easily guessed passwords. (Motivation: [5, Ch.3].)

UTILITY OF ENTROPY AND GUESSWORK. For user-chosen passwords in practice, entropy and guesswork are of limited use, due to: (1) the difficulty acquiring (even reliably estimating) password distribution probabilities, and (2) these metrics do not accurately model guessing attacks (see below). Almost all of the n possible passwords P_i in a space $\mathcal{X}$ never appear in a dataset, but their probability remains non-zero. P_i's frequency of appearance across m users in a sample should not be confused with q_i; a password dataset, and statistics computed from it, cannot be extrapolated to accurately estimate a probability distribution D_X. Determining which P_i are popular is still useful, e.g., for blacklisting too-popular passwords (see *pro-active password checking* in this chapter).

The sound practice of systems storing passwords hashed (vs. cleartext) complicates the study of real-world password distributions. Most analysis to date has been on large "leaked" (stolen, then published) datasets. Clever mechanisms have also been devised by which statistics can be collected without human access to cleartext password databases.[17] This has increased our knowledge, especially about passwords "at the head" of distributions, i.e., those of highest probability.[18] But probability distributions D_X for password spaces remain largely unknown. Consider a password allowable by system policy, but that has never shown up in an observed dataset. How should its probability be estimated? Models have been proposed, but their accuracy is difficult to judge.

WHICH METRICS ARE USEFUL IN PRACTICE. For equi-probable event spaces, such as randomly chosen cryptographic keys and system-assigned random passwords,

[16] G_1's sum in (3.7) assigns, to each event in $\mathcal{X}$, a guess-charge i weighted by a probability q_i, with optimal-order dictating $q_i \geq q_{i+1}$. For the 32 elements in $\mathcal{S}$ alone, we expect to need $2^5 \cdot 2^{127} = 2^{132}$ guesses; but if the average guess-charge for each of the sum's first m terms were 2^{107}, the guesswork component for these $m < n$ terms would be $m \cdot 2^{107} = 2^{132}$. All terms in the sum are non-negative. It follows that $G_1(\mathcal{R}) > 2^{107}$.

[17] See Bonneau [6].

[18] For example, see Weir [55].

Shannon entropy $H(X)$ is a useful metric; however its use for passwords too often results in the false assumption that user-chosen passwords are random and that attacks proceed by uninformed exhaustive search. Smarter attacks use tools that try passwords in priority order, i.e., (estimated or heuristic) highest probability first.

$H(X)$ can't measure strength of single passwords—as a sum over all possible events, it gives an average over the full event space. Thus $H(X)$ does not model real attacks which guess in priority order over a subset of the password space (the head of the distribution), "quitting early" to avoid expending effort on the hardest passwords. Guesswork is similarly a full-distribution average—consequently (example above), it is subject to being distorted by outliers. Thus both $\mathrm{H}(X)$ and guesswork poorly measure real-world attacks. Min-entropy provides useful single-point information, albeit coarse (relative to only a single-guess)—but it helps move us towards *partial guessing metrics*.

PARTIAL GUESSING METRICS. As noted above, real-world guessing attacks often guess through only partial password spaces, selectively. Thus metrics which average over full event spaces are poor models. Measures that better reflect the strategies behind real attacks include two *partial guessing metrics* we now mention. The β-*success rate* function

$$B_{\mathcal{X}}(\beta) = \sum_{i=1}^{\beta} q_i \tag{3.9}$$

gives the expected success probability for an attack capable of β guessing trials on an account from password space $\mathcal{X}$. To explain $B_{\mathcal{X}}(10,000) = 0.20$, suppose a systems administrator finds, from analysis of the company password database, that the most popular 10,000 distinct passwords are used by 20% of its users: $q_1 + q_2 + ... + q_{10,000} = 0.20$. Then an attacker able to mount 10,000 guesses on one account (randomly chosen from all accounts) would guess the correct password with probability 20%, or is expected to break into 20% of accounts if given 10,000 guesses on each account. This would correspond to an online attack on a system that did not rate-limit (throttle) login attempts.

For probabilities q_i from a password space $\mathcal{X}$, the α-*work factor* function

$$A_{\mathcal{X}}(\alpha) = min\{a \mid \sum_{i=1}^{a} q_i \geq \alpha\} \tag{3.10}$$

gives the number a of per-account guesses to correctly guess proportion α of accounts, or for one account to correctly guess its password with probability at least α; or correspondingly, the cardinality of an optimal dictionary to do so. If the parameter $\alpha = 0.20$ or 20% is chosen, this metric tells us what number of per-account guesses is expected to be needed to guess 20% of accounts (or break into one account, drawn randomly from system accounts, with probability 20%). For the scenario immediately above, this metric would return the answer $a = 10,000$.

3.10 ‡Endnotes and further reading

The 1985 "Green Book" [17] (rainbow series) discusses system-generated random passwords; on password aging it recommends a maximum password lifetime of one year. Many Unix systems support password aging. NIST 800-63-2 [9] discusses Equation (3.1), setting a limit of 100 login attempts over 30 days, and (to avoid lock-out) still accepting login attempts from white-listed IP addresses from which successful logins previously occurred. Secret salts are discussed by Manber [35] and Abadi [1]; attacks on secret salt (but not on iterated hashing) can be parallelized. Oechslin's *rainbow tables* [44] allow an advanced attack on unsalted passwords, using a time-memory tradeoff; see also Narayanan [42]. Provos [47] explains a future-adaptable backwards-compatible hashing scheme whereby a hash iteration count can be increased over time as computing power increases. Florêncio [22] summarizes password research for systems administrators. Hatzivasilis [30] provides an overview of the 2013-2015 Password Hashing Competition candidates and winner Argon2; Dürmuth [18] explores offline guessing attacks leveraging parallel computing architectures for fast hashing, including GPUs.

From a large-scale empirical study, Bonneau [7] concludes: "it appears next to impossible to find secret questions that are both secure and memorable". The password reset attack is from Gelernter [26]. The description of passcode generators and Lamport's OTP chain is adapted from Menezes [38]; see also Bellcore's S/KEY one-time password system [27, 28]. Bonneau [8] discusses comparative analysis of password alternatives, including hardware tokens, biometrics, password managers, and secret questions.

For usability of password managers see Chiasson [13]; Florêncio [23] explores managing *password portfolios*. Biddle [4] surveys graphical passwords. For CAPTCHAs, Hidalgo [31] gives an authoritative survey; Motoyama [40] explores underground CAPTCHA-solving economies; Egele [19] discusses CAPTCHA farms that leverage malware. Van Oorschot [53] extends the usability of the Pinkas-Sander protocol [46] by tracking failed login counts. Garfinkel [24] surveys usable security work, especially authentication. For *password meters*, see de Carné de Carnavalet [15] and Wheeler's zxcvbn [56].

For authoritative surveys on biometrics, see Jain [33, 32]. For effective attacks on fingerprint systems, including using "gummy bears" (gelatin), see Matsumoto [36]. For iris recognition, see Daugman [14, 29]. Ballard [3] notes "the evaluation of biometric technologies is usually conducted under fairly weak adversarial conditions", encourages use of more realistic attack models (e.g., "trained" forgers), and demonstrates successful impersonation attacks on handwriting (a behavioural biometric) using *generative attacks* that synthesize forged inputs. For attacks on touch-screen implicit authentication schemes (another behavioural biometric), see Khan [34]. For background on keystroke dynamics (latencies), see Monrose [39]. For using biometrics for cryptographic key generation, see Ballard [2]. For a primer on secure Internet geolocation, see Muir [41].

Figure 3.6 follows Shannon [50, p.20], who introduces entropy and succinctly derives basic properties; see also McEliece [37], Garrett [25], Bonneau [5, Ch.3]; Ferguson-Schneier [21] and Cachin [11]. *Weak password subspaces* [54] are related to partial guessing metrics and predictable user-choices. Heuristics for estimating password strength

based on Shannon entropy, popularized by SP 800-63 [10, Appendix A], were widely used to justify password composition policies despite explicit warning therein that they were very rough rules-of-thumb; the revision 13 years later [45, Part B], mentioned in Section 3.2, recommends against not only password aging (following [57, 12]), but also against composition policies after Weir [55] and others [6] explained how poorly the heuristic models match practical attack strategies.

References

[1] M. Abadi, T. M. A. Lomas, and R. Needham. Strengthening passwords. SRC Technical Note 1997-033, DEC Systems Research Center, Palo Alto, CA, 1997. September 4 with minor revision December 16.

[2] L. Ballard, S. Kamara, F. Monrose, and M. K. Reiter. Towards practical biometric key generation with randomized biometric templates. In *ACM Conf. Computer and Comm. Security (CCS)*, pages 235–244, 2008.

[3] L. Ballard, F. Monrose, and D. P. Lopresti. Biometric authentication revisited: Understanding the impact of wolves in sheep's clothing. In *USENIX Security Symp.*, 2006.

[4] R. Biddle, S. Chiasson, and P. C. van Oorschot. Graphical passwords: Learning from the first twelve years. *ACM Computing Surveys*, 44(4):19:1–19:41, 2012.

[5] J. Bonneau. *Guessing Human-Chosen Secrets*. PhD thesis, University of Cambridge, U.K., 2012.

[6] J. Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *IEEE Symp. Security and Privacy*, pages 538–552, 2012.

[7] J. Bonneau, E. Bursztein, I. Caron, R. Jackson, and M. Williamson. Secrets, lies, and account recovery: Lessons from the use of personal knowledge questions at Google. In *WWW—International Conf. on World Wide Web*, pages 141–150, 2015.

[8] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *IEEE Symp. Security and Privacy*, pages 553–567, 2012.

[9] W. E. Burr, D. F. Dodson, E. M. Newton, R. A. Perlner, W. T. Polk, S. Gupta, and E. A. Nabbus. NIST Special Pub 800-63-1: Electronic Authentication Guideline. U.S. Dept. of Commerce. Dec 2011 (121 pages), supersedes [10]; superseded by SP 800-63-2, Aug 2013 (123 pages), itself superseded by [45].

[10] W. E. Burr, D. F. Dodson, and W. T. Polk. NIST Special Pub 800-63: Electronic Authentication Guideline. U.S. Dept. of Commerce. Ver. 1.0, Jun 2004 (53 pages), including Appendix A: Estimating Password Entropy and Strength (8 pages). Superseded by [9].

[11] C. Cachin. *Entropy Measures and Unconditional Security in Cryptography*. PhD thesis, Swiss Federal Institute of Technology Zurich, Switzerland, May 1997.

[12] S. Chiasson and P. C. van Oorschot. Quantifying the security advantage of password expiration policies. *Designs, Codes and Cryptography*, 77(2-3):401–408, 2015.

[13] S. Chiasson, P. C. van Oorschot, and R. Biddle. A usability study and critique of two password managers. In *USENIX Security Symp.*, 2006.

[14] J. Daugman. How iris recognition works. *IEEE Trans. Circuits Syst. Video Techn.*, 14(1):21–30, 2004.

[15] X. de Carné de Carnavalet and M. Mannan. A large-scale evaluation of high-impact password strength meters. *ACM Trans. on Information Systems and Security*, 18(1):1:1–1:32, 2015.

[16] P. J. Denning, editor. *Computers Under Attack: Intruders, Worms, and Viruses*. Addison-Wesley, 1990. Edited collection (classic papers, articles of historic or tutorial value.

[17] DoD. Password Management Guideline. Technical Report CSC-STD-002-85 (Green Book), U.S. Department of Defense. 12 April 1985.

[18] M. Dürmuth and T. Kranz. On password guessing with GPUs and FPGAs. In *PASSWORDS 2014*, pages 19–38.

[19] M. Egele, L. Bilge, E. Kirda, and C. Kruegel. CAPTCHA smuggling: hijacking web browsing sessions to create CAPTCHA farms. In *ACM Symp. Applied Computing (SAC)*, pages 1865–1870, 2010.

[20] M. W. Eichin and J. A. Rochlis. With microscope and tweezers: an analysis of the Internet virus of November 1988. In *IEEE Symp. Security and Privacy*, pages 326–343, 1989.

[21] N. Ferguson and B. Schneier. *Practical Cryptography*. Wiley Publishing, 2003.

[22] D. Florêncio, C. Herley, and P. C. van Oorschot. An administrator's guide to Internet password research. In *Large Installation System Administration Conference (LISA)*, pages 35–52. USENIX, 2014.

[23] D. Florêncio, C. Herley, and P. C. van Oorschot. Password portfolios and the finite-effort user: sustainably managing large numbers of accounts. In *USENIX Security Symp.*, pages 575–590, 2014.

[24] S. L. Garfinkel and H. R. Lipford. *Usable Security: History, Themes, and Challenges*. Synthesis Lectures on Information Security, Privacy, and Trust. Morgan & Claypool Publishers, 2014.

[25] P. Garrett. *The Mathematics of Coding Theory*. Pearson Prentice Hall, 2004.

[26] N. Gelernter, S. Kalma, B. Magnezi, and H. Porcilan. The password reset MitM attack. In *IEEE Symp. Security and Privacy*, pages 251–267, 2017.

[27] N. Haller. The S/KEY One-Time Password System. In *ISOC Symp. Network and Distributed System Security (NDSS)*, 1994.

[28] N. Haller and C. Metz. RFC 1938: A one-time password system, May 1996. Cf. RFC 1760 (Feb 1995).

[29] F. Hao, R. J. Anderson, and J. Daugman. Combining crypto with biometrics effectively. *IEEE Trans. Computers*, 55(9):1081–1088, 2006.

[30] G. Hatzivasilis. Password-hashing status. *Cryptography*, 1(2):10:1–10:31, 2017.

[31] J. M. G. Hidalgo and G. Á. Marañón. CAPTCHAs: An artificial intelligence application to web security. *Advances in Computers*, 83:109–181, 2011.

[32] A. K. Jain, A. Ross, and S. Pankanti. Biometrics: a tool for information security. *IEEE Trans. Info. Forensics and Security*, 1(2):125–143, 2006.

[33] A. K. Jain, A. Ross, and S. Prabhakar. An introduction to biometric recognition. *IEEE Trans. Circuits Syst. Video Techn.*, 14(1):4–20, 2004.

[34] H. Khan, U. Hengartner, and D. Vogel. Targeted mimicry attacks on touch input based implicit authentication schemes. In *MobiSys 2016 (Mobile Systems, Applications, and Services)*, pages 387–398, 2016.

[35] U. Manber. A simple scheme to make passwords based on one-way functions much harder to crack. *Computers & Security*, 15(2):171–176, 1996.

[36] T. Matsumoto, H. Matsumoto, K. Yamada, and S. Hoshino. Impact of artificial "gummy" fingers on fingerprint systems. In *Proc. SPIE 4677, Optical Security and Counterfeit Deterrence Techniques IV*, pages 275–289, 2002.

[37] R. J. McEliece. The Theory of Information and Coding. In G.-C. Rota, editor, *Encyclopedia of Mathematics and Its Applications*, volume 3. Addison-Wesley, Reading, MA, 1977.

[38] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. Freely available online.

[39] F. Monrose, M. K. Reiter, and S. Wetzel. Password hardening based on keystroke dynamics. *Int. J. Inf. Sec.*, 1(2):69–83, 2002.

[40] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re:CAPTCHAs—Understanding CAPTCHA-solving services in an economic context. In *USENIX Security Symp.*, pages 435–462, 2010.

[41] J. A. Muir and P. C. van Oorschot. Internet geolocation: Evasion and counterevasion. *ACM Computing Surveys*, 42(1):4:1–4:23, 2009.

[42] A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *ACM Conf. Computer and Comm. Security (CCS)*, pages 364–372, 2005.

[43] NIST. FIPS 112: Password Usage. National Inst. Standards and Tech., U.S. Dept. of Commerce, May 1985.

[44] P. Oechslin. Making a faster cryptanalytic time-memory trade-off. In *CRYPTO—Advances in Cryptology*, pages 617–630, 2003.

[45] Paul A. Grassi et al. NIST Special Pub 800-63-3: Digital Identity Guidelines. U.S. Dept. of Commerce. Jun 2017, supersedes [9]. Additional parts SP 800-63A: Enrollment and Identity Proofing, SP 800-63B: Authentication and Lifecycle Management, SP 800-63C: Federation and Assertions.

[46] B. Pinkas and T. Sander. Securing passwords against dictionary attacks. In *ACM Conf. Computer and Comm. Security (CCS)*, pages 161–170, 2002.

[47] N. Provos and D. Mazières. A future-adaptable password scheme. In *USENIX Annual Technical Conf.*, pages 81–91, 1999. FREENIX Track.

[48] J. A. Rochlis and M. W. Eichin. With microscope and tweezers: the Worm from MIT's perspective. *Commun. ACM*, 32(6):689–698, 1989. Reprinted in [16] as Article 11; a more technical related paper by these authors appeared as [20].

[49] A. D. Rubin. *White-Hat Security Arsenal*. Addison-Wesley, 2001.

[50] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, vol.27, 1948. Pages 379–423 (Jul) and 623–656 (Oct).

[51] E. H. Spafford. Crisis and aftermath. *Commun. ACM*, 32(6):678–687, 1989. Reprinted in [16] as Article 12.

[52] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay. Measuring real-world accuracies and biases in modeling password guessability. In *USENIX Security Symp.*, pages 463–481, 2015.

[53] P. C. van Oorschot and S. G. Stubblebine. On countering online dictionary attacks with login histories and humans-in-the-loop. *ACM Trans. on Information Systems and Security*, 9(3):235–258, 2006.

[54] P. C. van Oorschot and J. Thorpe. On predictive models and user-drawn graphical passwords. *ACM Trans. on Information Systems and Security*, 10(4):1–33 (Article 17), 2008.

[55] M. Weir, S. Aggarwal, M. P. Collins, and H. Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *ACM Conf. Computer and Comm. Security (CCS)*, pages 162–175, 2010.

[56] D. L. Wheeler. zxcvbn: Low-budget password strength estimation. In *USENIX Security Symp.*, pages 157–173, 2016.

[57] Y. Zhang, F. Monrose, and M. K. Reiter. The security of modern password expiration: an algorithmic framework and empirical analysis. In *ACM Conf. Computer and Comm. Security (CCS)*, pages 176–186, 2010.