New Paradigms for Password Security
(abstract from the keynote lecture)

Xavier Boyen
xb@boyen.org

Voltage Inc.

For the past several decades, cryptographers have consitently provided us with stronger and more capable primitives and protocols that have found many applications in security systems in everyday life. One of the central tenets of cryptographic design is that, whereas a system's architecture ought to be public and open to scrutiny, the keys on which it depends — long, utterly random, unique strings of bits — will be perfectly preserved by their owner, and yet nominally inaccessible to foes.

This security model works well as long as one can assume the existence of an inviolate physical location or storage device to safeguard those keys. In client-server scenarios, the mere delocalization of the participants suffices to enforce a proper boundary without any further precaution. In proxy settings, one may call upon tamper-resistant "smart cards" or hardware security modules to isolate the keys adequately from most opponents.

Things break down when one can no longer assume that an external storage medium is available to store our keys, and that the only option is to remember them in our minds. The problem, of course, is a cognitive one: the human brain is ill-equipped to remember hundreds of random bits of key material for the long term without making any mistake. The secrets that our brain is keen on remembering are those of our own choosing, which for all their apparent randomness and unpredictability can certainly not be mistaken nor substituted for genuine cryptographic keys. Security from purely mental secrets requires us at the very least to compromise on key strength — this encompassing both entropy and uniformity —, and seek the best reachable security goals based not on ideal random keys but on passwords of sub-cryptographic quality.

Plain textual passwords and passphrases — or passtexts — have always been the preferred form of human-memorable secret, having the benefit of medium-independence which entails compatibility with virtually any conceivable user interface. More exotic mental secrets — passthoughts — may be based on visual or auditory recognition feedback; these are equivalent to passwords from a cryptographic perspective, but the specialized input device they require make them less practical. Secrets whose expression requires body action such as speech or ocular movements — passmoves — may also be envisaged given the proper measurement apparatus, with the proviso that the unavoidable measurement noise in the analog signal will have to be dealt with; we merely mention that errors on the post-quantization signal may be correctable using information-theoretic cryptographic tools such as reusable and robust fuzzy extractors [1] without leaking excessive information about the secret.

Regardless of the shape of form of the secret, an important criterion for its human memorability is that its selection ultimately be left to the human who will have to remember it. Machines can assist in password selection, but should not make the final choice. Because of this, it is a near-certainty that the selected secret will not make a suitable cryptographic key, nor will it be possible to derive one from it due to lack of entropy. Hence, specialized primitives and protocols are needed that explicitly take into account those inherent weaknesses, and seek to achieve the best possible security under the circumstances.

Although password-based primitives and protocols have seen much foundational and implementational improvements during the last two decades, the general philosophy of password-based offline key derivation and online key exchange has remained essentially what it was in the early nineties. In particular, most current approaches could better handle real-life situations where the password are too weak for comfort and/or are recycled in part or in whole with multiple correspondents.

The purpose of this exposé is thus to investigate what security may indeed be attained from human-memorable passwords as they do appear in the real world — including the weak, skewed, reused, and exceedingly long-lived ones. The focus on literal passwords stems from tradition as much as convenience.

1 Halting Puzzles against Brute-Force Dictionary Attacks

Stand-alone — offline — uses of passwords mainly concern encryption and key derivation applications. The prime example of this is to encrypt the contents of a laptop so that only its owner can access it. Local authentication and device unlocking uses may also be treated as special cases of password-based encryption. At the core of these systems, one finds a Key Derivation Function (KDF), which is a one-way function taking a password and an optional public random salt as input, and producing a reproducible cryptographic key as output.

Offline applications such as those are tremendously difficult to secure with a weak password. The threat model here is the loss of the entire ciphertext and all associated hardware to an attacker, where only the password is being held back. Therefore, any opponent that simply tries out all passwords in an offline dictionary attack, e.g., by decreasing order of estimated likelihood, will eventually stumble upon the correct one and defeat the encryption. The only defense against such a threat is to slow down or deter the attacker by making the attack more daunting. There are two ways to do this: by picking an unlikely password to increase the expected number of guesses, and by making each guess more computationally demanding to verify.

One cannot really make assumptions about the choice of password, only encourage the user to pick one that is long and difficult to guess. Making the guesses hard to verify is possible, but only within limits, as it has the side effect of increasing the user's legitimate access latency in the same proportion. For this reason, KDFs are purposely designed to be somewhat expensive to compute, although most implementations tend to be very conservative with the amount of slowdown that they are willing to impose on users, and rarely offer the user any choice in the matter. The general trend is thus to use KDFs with a slowdown parameter (often a hash iteration count) that is conservatively chosen, forever frozen, and publicly disclosed as part of the KDF specification or implementation. Some implementations support in-the-field adjustment of the KDF iteration count, but this parameter always remains public.

This has been and continues to be the ubiquitous way in which passwords are used for local key derivation.

In departure from this trend, we recently introduced, in [2], the notion of Halting Key Derivation Function (HKDF), which explicitly lets the user choose an arbitrary hardness parameter and embed it into the function in a cryptographically secret manner. The idea is to encourage the user to make the HKDF as difficult to compute as the delay he or she is willing to tolerate when seeking access, but conceal the value of the chosen parameter from public view, and yet not require the user to remember such value — or for that matter anything else besides the password.

The crucial element is that, on the correct password, the HKDF function will recognize that it succeeded and halt spontaneously after the intended computational delay; but on an incorrect password, it will continue indefinitely without giving any feedback until manually interrupted. The only indication given to the user that a password is incorrect will be the feeling that the key derivation is taking longer than it should. The user will naturally react by restarting the process and reentering the password more carefully without much of an afterthought. To an attacker, by contrast, this lack of feedback will disproportionately complicate the task of mounting an offline dictionary attack. The result is an effective security increase equivalent to two extra bits of password entropy, at virtually no cost to the legitimate user.

The total security gain provided by HKDFs is actually much greater than those two bits, due to a combination of factors. The main contributing factor is that legacy KDFs tend to be parameterized very conservatively, leading to exceedingly short delays (~ 1 *ms*) that are only getting shorter as computers are getting faster, raising obsolescence concerns. By contrast, HKDFs are programmed on a case-by-case basis by a user who perceives actual clock times. Even at the shorter end of user latencies, a "blink of an eye" slowdown (~ 1 *sec*) already results in a substantial security jump on today's machines. The boost will also keep up with technological progress, since a one-second delay set in ten years will entail more processor cycles than a one-second delay set today.

As discussed in [2], one should expect a fairly wide spectrum of user-selected HKDF delays to find their way into practical territory. Short delays are appropriate for frequently used day-to-day passwords with a short lifespan. Longer delays (~ 1 *min* and more) can be used to protect longer-term backup passwords, which must be easier to remember over a longer period, but can tolerate the occasional slowdown. The longest delays (~ 1 *hour* and more) would be reserved for lastresort disaster-recovery passwords, never intended to be used, but that must be available and remembered should the need ever arise, even after many years have lapsed. Such passwords will likely be fairly weak to be reliably memorable over extended periods, whilst most of the protection from offline dictionary attacks is ensured by the very long HKDF cryptographic delays attached thereto.

Notably, the same plaintext may be seamlessly HKDF-encrypted under many such passwords with widely different delays, without loss of security or usability.

2 Hardened Protocols toward Universal Authentication

Client-server — online — uses of passwords are primarily geared toward authentication and key exchange. Both parties share a password, and, based on it, try to establish a private authenticated channel over open communication lines. The constraints on online passwords are fairly different than in the offline case, as here the threat model typically assumes that the communicating parties are honest and try to prevent eavesdropping and impersonation by a malicious outsider (who controls the underlying communication channel).

Password-Authenticated Key Exchange (PAKE) is indeed a success story of cryptographic protocol design, as there are many protocols realizing the theoretically optimal security requirement that the only feasible attack vector be for the adversary to make online password guesses, one guess at a time, interactively with one of the honest parties — who can then detect the attack and throttle it by refusing to communicate. Secure online authentication can thus be achieved using much weaker passwords than would be thinkable in the offline case.

Extensions of this notion have been proposed for the case where the server itself may be viewed as an adversary, as is the case when the client wishes to reuse the same password with other servers. Asymmetric Password-Authenticated Key Exchange (APAKE) deals with this notion by requiring the password only on the client side; the server is instead entrusted with a derived secret that can be used to reach mutual authentication with the client, but not impersonate it to another server (in particular the password should be hard to recover from this). APAKE protocols are for this reason more desirable in practical use than PAKE, in light of the well-documented propensity of internet users to recycle the same few passwords with a broad variety of vendors. However, one concern remains, which is how difficult it actually is for a malicious server to recover its clients' passwords from the derived secrets.

The concern is that the derived secrets are typically obtained by applying a one-way function to the password w, be it a cryptographic hash $h(w)$ or a modular exponentiation g^w. Functions like these are usually very fast to compute, so even though they technically may be one-way, they might be relatively easy to invert in an offline dictionary attack if the user password is not already very strong. Also, without an extra randomization step, a server can attack all of its clients' passwords for the price of one.

Since typical real-life users are probably going to continue reusing the same weak passwords with many servers regardless of whether this is considered a safe thing to do, it would be desirable to design a protocol that attempts to preserve the best possible form of online and offline password security, even under reuse of a weak password across multiple servers. The benefit from such a notion would be safe universal authentication on the internet using a single easy-to-remember password (for each user).

Ideally, one wishes to combine the security of (A)PAKE against outside online attackers, with the security of HKDF against malicious servers.

To this end, we are proposing, in [3], the notion of Hardened Password-Authenticated Key Exchange (HPAKE), which offers security guarantees comparable to those of regular asymmetric password key exchange, and in addition allows the user to specify an arbitrarily expensive one-way function à la HKDF for mapping the client password to the server secret. This renders even relatively weak passwords infeasible to recover by malicious servers, thereby enabling the reuse of such passwords with arbitrarily many servers.

There are several difficulties with this. The first is a systemic one: the burden of computing this arbitrarily expensive one-way function should befall the client who selected it, and not the server which for scalability reasons must be able to process many simultaneous authentication requests with minimal effort. The second issue is technical: since the one-way function is to be computed on the client side, the client must obtain the necessary inputs from the server prior to authentication. This creates a paradox, since the success of such transfer must depend on the client's knowledge of the password, but at the same time not directly reveal to either the client or the server whether the transfer succeeded, lets it open an avenue for offline attack to outsiders or to the server itself.

We shall discuss how these difficulties can be overcome, and how the HPAKE framework from [3] provides a plausible and practical answer to the problem of universal authentication from a single password.

Conclusion

The password schemes presented in this lecture have in common that they seek to provide the best possible security for the password holder, in the offline and online setting, regardless of how careless his or her use of that password may be. The only safety rule that should never be failed, is that one's password should only be seized on a local trusted HKDF or HPAKE entry device, and not shared with other less secure protocols.

References

1. Boyen, X.: Robust and Reusable Fuzzy Extractors. In: Tuyls P., Skoric B., Kevenaar T. (eds.), Security with Noisy Data. Springer-Verlag (2007)
2. ——: Halting Password Puzzles – hard-to-break encryption from human-memorable keys. In: SECURITY 2007, The USENIX Association (2007)
3. ——: Hardened Password Authentication – mulitple mobile credentials from a single short secret. Manuscript (2008)