Editors: Peter Gutmann, pgut001@cs.auckland.ac.nz | David Naccache, david.naccache@ens.fr | Charles C. Palmer, ccpalmer@us.ibm.com

Honey Encryption

Encryption beyond the Brute-Force Barrier

Ari Juels | Cornell Tech
Thomas Ristenpart | University of Wisconsin

What do the strings 123456, 12345, 123456789, 1234567, and 12345678 have in common? Apart from obvious lexical similarities, they're among the most popular passwords selected by ordinary users. Among some 32 million famously leaked from the popular RockYou online service in 2009,[1] these strings constitute—in order of popularity—five of the 10 most common user passwords. The most common password was 123456, chosen by nearly 1 percent of RockYou's users.

Users' all-too-common selection of passwords that are easy to guess has been the subject of much recent research[2,3] and hand-wringing by security administrators.

The problem of weak passwords prompted us to introduce *honey encryption* (HE).[4] (In computer security, *honey* commonly denotes a false resource designed to lure or deceive an attacker. Honeypots, for example, are servers designed to attract attackers for observation and study.[5]) HE creates a ciphertext that, when decrypted with an incorrect key or password, yields a valid-looking but bogus message. So, attackers can't tell when decryption has been successful.

The Vulnerability of Weak Passwords

Weak passwords are vulnerable to powerful offline password-cracking attacks. Passwords are often protected in server databases through a one-way cryptographic function h called a *hash function*. For example, user Alice's password P is stored in its hashed form $h(P)$. (To prevent adversarial precomputation of hashes over passwords, it's common practice to incorporate a *salt*, a password-specific random value s, into the hash—that is, to store $h(P, s)$.) A password P' proffered by a user trying to log in as Alice can be verified through fresh hashing and comparison with the stored hash.

Hash-Cracking Tools

Hacking tools such as John the Ripper (www.openwall.com/john) enable even mere hobbyists to easily crack these hashes when the underlying passwords are weak. These tools exploit knowledge of how users typically compose their passwords, thanks to compromises such as the RockYou breach. They generate guesses according to passwords' conjectured popularity. They're why recent breaches involving the disclosure of hashed passwords—for example, the 2014 breach of Yahoo[6]—have been so worrisome.

Password-Based Encryption

Weak passwords aren't just a problem for hashing; they also impact users' ability to encrypt sensitive data using *password-based encryption* (PBE). PBE carries essentially the same vulnerability to guessing attacks as hashing.

PBE basics. A PBE scheme consists of an encryption function `enc` and a corresponding decryption function `dec`. A message M is encrypted under a password P as a ciphertext $C = \texttt{enc}_P(M)$. The message can be decrypted as $M = \texttt{dec}_P(C)$. Given a decryption attempt using an incorrect password $\tilde{P} \neq P$, $\texttt{dec}_{\tilde{P}}(C)$ outputs an error message—which we can think of for convenience as a special error symbol $\perp$. A popular standard for PBE is PKCS (Public-Key Cryptography Standard) #5 v2.0.[7] (Practitioners often use the key-derivation function PBKDF2 from PKCS #5 v2.0 to derive encryption or decryption keys from passwords. However, advances in cipher and cipher-mode design since the publication of PKCS #5 v2.0 motivate the use of authenticated encryption modes such as EAX over PKCS #5 v2.0 recommendations.)

Explicitly flagging decryption failures using $\perp$, and thus authenti-

1540-7993/14/$31.00 © 2014 IEEE

Copublished by the IEEE Computer and Reliability Societies

```
HEnc($K, M$)
$S \leftarrow\$ \texttt{encode}(M)$
$R \leftarrow\$ \{0, 1\}^n$
$S' \leftarrow H(R, K)$
$C \leftarrow S' \oplus S$
return $(R, C)$
(a)

HDec($K, (R, C)$)
$S' \leftarrow H(R, K)$
$S \leftarrow C \oplus S'$
$M \leftarrow \texttt{decode}(S)$
return $M$
(b)
```

Figure 1. A simple instantiation of DTE-then-encrypt (DTE stands for distribution-transforming encoder). (a) Honey encryption. (b) Honey decryption. H is a cryptographic hash function, K is a key, M is a message, S is a seed, R is a random string, C is a ciphertext, and $\leftarrow\$$ denotes uniform random assignment.

cating PBE ciphertexts, alerts users to typos or misremembered passwords. But authenticated encryption also means that adversaries making password-guess attempts against a PBE ciphertext know when they've decrypted successfully. A PBE ciphertext is therefore secure only up to the attacker's ability to guess P through brute force, a fundamental security limitation we call the *brute-force bound*.

Common hash-cracking tools can be nicely repurposed to exploit the brute-force bound. They give PBE attackers exactly what they need: a list of passwords to try against a ciphertext, generated in order of their (presumed) popularity with users.

Why care about PBE? PBE is used to protect many forms of sensitive user data and is used notably in *password managers*.

The recent litany of password breaches, along with improved cracking tools such as John the Ripper, hasn't escaped consumers' notice. Many users store and protect their passwords in password managers, such as DashLane, LastPass, or Apple's iCloud keychain. Password managers let users compile a small database of passwords and their associated accounts. They can store existing, human-selected passwords and optionally generate strong passwords—ones with effectively uncrackable hashes. They store passwords—not their hashes—in a database M, treat it as a plaintext message, and then encrypt it under a user-selected master password P^*. So, password managers are vulnerable to the brute-force cracking of P^*.

This is a problem because individual users' encrypted password databases are often stored by password manager services in the cloud. If such a service is breached, these databases become vulnerable to offline PBE cracking. (LastPass reported an apparent breach and leakage of some users' password databases in 2011.[8]) This threat is serious because a cracked database frequently contains all of a user's Web passwords.

Similarly, private keys used in SSH (Secure Shell), such as RSA private keys, can be PBE-encrypted for protection on a client. These too are vulnerable to brute-force guessing attacks.

Introducing HE

For some kinds of messages, HE provides security that's beyond the brute-force bound and thus unobtainable with traditional PBE. HE constructs a ciphertext C that decrypts under any password to a plausible-looking message. Under HE, $\tilde{P}$ yields a fake message $\tilde{M}$ that looks as valid to an attacker as M. As you'll see, our HE schemes accomplish this by first applying a specialized encoding mechanism and then encrypting the result with a conventional password-based encryption scheme. Our approach is conceptually similar to, but technically different from, compression followed by encryption.

Suppose Alice's password manager database is encrypted with HE and decrypts under an incorrect master password $\tilde{P}^*$ to yield a list of fake passwords or accounts. An attacker who tries these passwords online will fail to impersonate Alice.

HE's security properties might seem counterintuitive. With traditional PBE, an unbounded attacker, one who can make an unlimited number of guesses against (a bounded-length) P, can crack C with 100-percent success. Surprisingly, against an HE ciphertext, such an adversary can't successfully decrypt C with certainty, even if P is weak.

A Naive HE Attempt: The One-Time Pad

A classic cipher with security against an unbounded adversary is the *one-time pad*. This cipher expresses M as an ℓ-bit string ($M \in \{0, 1\}^{\ell}$). It selects an ℓ-bit key K uniformly at random ($K \leftarrow\$ \{0, 1\}^{\ell}$). (Here, $\leftarrow\$$ denotes uniform random assignment.) The message and key are combined through bitwise XOR as $C = M \oplus K$, so that the key masks the message.

Given C, every possible ℓ-bit message $\tilde{M}$ is an equally plausible plaintext because a key $\tilde{K}$ exists such that $C = \tilde{M} \oplus \tilde{K}$. Also, every key is equally likely to have been selected at random. One-time pads thus achieve perfect information-theoretic or Shannon security.[9]

But the one-time pad isn't an HE scheme, for two reasons. First, K must be the same length as M and thus equivalent to a very strong (high-entropy) password. HE should work even with weak passwords. Second, with the one-time pad, most keys fail to yield plausible plaintexts and instead correspond to random strings. An adversary who knows, for instance, that M is an English-language message can discount such keys.

Distribution-Transforming Encoders

The one-time pad operates over general bit strings, without sensitivity to message formats or properties. HE instead models the space of plaintext messages using a *distribution-transforming encoder* (DTE).

Let p_m be a probability distribution over the message space $\mathcal{M}$, meaning that a user selects $M \in \mathcal{M}$ for encryption with probability $p_m(M)$. (We give an example for this intuition later.)

A DTE `encode` encodes M as an ℓ-bit *seed* $S \in \{0, 1\}^{\ell}$. This encoding needn't be unique: many seeds might correspond to M, in which case `encode` selects one such seed uniformly at random. (Every seed, however, corresponds to a unique message.)

We require that `encode` be efficiently invertible. In other words, given S, we can decode through the inverse DTE $\texttt{decode}(S) = M$, which returns S's unique corresponding message.

With a DTE that gives strong security (as we explain later), `decode` accurately generates p_m. In this case, selecting S uniformly at random from $\{0, 1\}^{\ell}$ and decoding to obtain $M = \texttt{decode}(S)$ returns approximately the original p_m. In other words, the DTE is a good model of the message distribution.

HE Construction

Given a good DTE, constructing a good HE scheme is simple using our *DTE-then-encrypt* approach. Recall that the goal is to encrypt M under K to yield C.

To apply HE to M, we encode it as $S \leftarrow\!\$\ \texttt{encode}(M)$ and encrypt S under K using suitable conventional symmetric-key encryption. For example, we can hash K as S' in the seed space and just pad the message with it. Figure 1 shows this scheme.

Figure 1 is for presentation purposes only. It doesn't reflect important PBE security mechanisms, such as those that substantially slow brute-force attacks when K has high entropy.[7] You can view HE's security guarantees as a hedge against the failure (due to low-entropy keys) of such conventional cryptographic protections—an example of defense-in-depth.

Security

HE aims to provide message-recovery security—that is, to prevent an adversary from determining M from C under K with high probability. As we mentioned before, this should hold even when the attacker is unbounded.

HE provides message-recovery security assuming that an adversary knows p_m and p_k, the distribution from which K is drawn. The adversary is presumed to know how users tend to select passwords.

In many settings of interest, we can show that given C, the best an adversary can do in guessing M is to decrypt C under the most probable K under p_k (for example, the password 123456 if K is a user-selected password). This is often equivalent to guessing the most probable M under p_m—no better than an adversary can do without the ciphertext.

For the formal security definitions for HE, see "Honey Encryption: Security beyond the Brute-Force Bound."[4]

An HE Example: Encrypting Ice Cream Flavors

A toy example helps convey more of the intuition behind HE. Suppose Bob wants to encrypt his favorite ice cream flavor M = "chocolate" (to send to Alice as a hint for his birthday) under a secret four-digit PIN K = 0000 (shared with Alice).

Bob looks up surveys on favorite ice cream flavors and finds that one-half of the respondents favored vanilla, one-fourth chose chocolate, and one-fourth preferred strawberry.

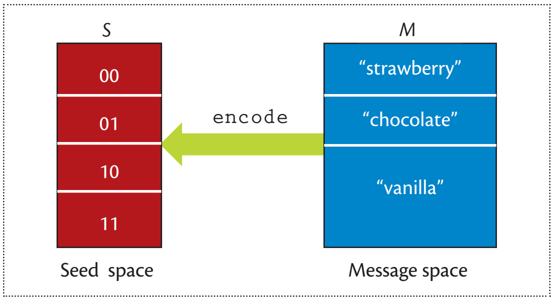

Figure 2. A DTE for favorite ice cream flavors. The message "strawberry" (with p_m = 1/4) maps to 00, "chocolate" (with p_m = 1/4) maps to 01, and "vanilla" (with p_m = 1/2) maps to {10, 11}. p_m is a probability distribution over the message space.

Bob thus constructs a favorite-ice-cream DTE that maps messages {"vanilla," "chocolate," "strawberry"} into the space of 2-bit strings {00, 01, 10, 11}. Figure 2 depicts a natural DTE for this purpose. (Note that `encode`("vanilla") picks either 10 or 11 at random with equal probability.) Encoding his message "chocolate," Bob obtains the seed `encode`("chocolate") = S = 01.

Bob selects a random string R and computes $S' = H(R, K)$. Suppose that $S' = H(R, 0000) = 11$. Then Bob computes $C = 11 \oplus 01 = 10$ and sends it to Alice.

Alice decrypts C by using her knowledge of the PIN to compute $S' = H(R, 0000) = 11$ and then $S = C \oplus S' = 10 \oplus 11 = 01$. Next, she computes $\texttt{decode}(01)$ = "chocolate" to recover Bob's message.

Suppose that Eve, an adversary, intercepts C and tries to decrypt it. She doesn't know Bob's PIN. She knows only that the most popular PIN is 1234. Now suppose that $H(R, 1234) = 00$. Eve computes $\tilde{S}' = H\left(R, 1234\right) = 00$ and then $\tilde{S} = C \oplus \tilde{S}' = 10$. Her best guess at the message is $\texttt{decode}(10)$ = "vanilla."

No matter what PIN Eve tries to decrypt with, the result will be a valid ice cream flavor. But with no further information, Eve can't do better than just guessing vanilla.

Another View of HE

You can think of a DTE with good message-recovery security as "smoothing" the distribution of plaintext messages. That is, for M chosen according to p_m, the output of $\texttt{encode}(M)$ should be close to the uniform distribution of ℓ-bit strings.

In this view, DTE-then-encrypt resembles compress-then-encrypt. A DTE is indeed conceptually similar to compression algorithms such as arithmetic coding.[10] However, important technical differences exist. For example, seeds don't need to be shorter than messages, so compression might not occur in HE.

Honey encryption introduces many interesting technical challenges. A good DTE is tailored to the message distribution over which encryption is taking place. Constructing a DTE for highly structured data, such as credit card numbers, is relatively straightforward but can otherwise be challenging. For example, constructing a DTE for the databases in password managers requires understanding how users select suites of passwords. Currently, researchers only have a good understanding of user selection of individual passwords (thanks to the RockYou breach). Fortunately, DTEs don't have to be perfect to provide useful security.

HE raises other interesting challenges. For instance, what happens if a user mistypes a password and decrypts a ciphertext (for example, an encrypted password database) into a wrong but valid-looking message? Several approaches to this typo-safety problem exist, such as associating different images or colors with different plaintexts, as in checking decrypted passwords online automatically.[11] Identifying the best approach remains an interesting open problem.

HE connects cryptography with distinctively different computer science domains, such as compression and natural-language processing. By applying the machinery of cryptography to the practice of decoys, HE promises to impart mathematical rigor to decoy and honey object use in system security. At the same time, HE translates the creative defensive tactic of decoys, traditionally the purview of system security, into cryptographic theory and practice. It thereby addresses a seemingly intractable problem—successfully encrypting messages using weak (low min-entropy) keys such as passwords against strong (unbounded) adversaries, and pitting human beings' limited memory against computing devices' vast power. HE represents cryptography's essence: it's at once useful and intriguingly counterintuitive. ■

References

1. M. Siegler, "One of the 32 Million with a RockYou Account? You May Want to Change All Your Passwords, Like Now," *TechCrunch*, 14 Dec. 2009; http://techcrunch.com/2009/12/14/rockyou-hacked.
2. J. Bonneau, "The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords," *Proc. 2012 IEEE Symp. Security and Privacy*, 2012, pp. 538–552.
3. M.L. Mazurek et al., "Measuring Password Guessability for an Entire University," *Proc. 2013 ACM Conf. Computer and Communications Security* (CCS 13), 2013, pp. 173–186.
4. A. Juels and T. Ristenpart, "Honey Encryption: Security beyond the Brute-Force Bound," *Advances in Cryptology—Eurocrypt 2014*, LNCS 8441, Springer, 2014, pp. 293–310; http://pages.cs.wisc.edu/~rist/papers/HoneyEncryptionpre.pdf.
5. L. Spitzner, "Honeytokens: The Other Honeypot," *Symantec SecurityFocus*, July 2003; www.symantec.com/connect/articles/honeytokens-other-honeypot.
6. J. Lyne, "Yahoo Hacked and How to Protect Your Passwords," *Forbes*, 31 Jan. 2014; www.forbes.com/sites/jameslyne/2014/01/31/yahoo-hacked-and-how-to-protect-your-passwords.
7. B. Kaliski, *PKCS #5: Password-Based Cryptography Specification Version 2.0*, IETF RFC 2898, Sept. 2000; http://tools.ietf.org/html/rfc2898.
8. L. Whitney, "LastPass CEO Reveals Details on Security Breach," *CNET*, 6 May 2011; www.cnet.com/news/lastpass-ceo-reveals-details-on-security-breach.
9. C.E. Shannon, "Communication Theory of Secrecy Systems," *Bell System Tech. J.*, vol. 28, no. 4, 1949, pp. 656–715.
10. I.H. Witten, R.M. Neal, and J.C. Cleary, "Arithmetic Coding for Data Compression," *Comm. ACM*, vol. 30, no. 6, 2007, pp. 520–530.
11. R. Dhamija and J.D. Tygar, "The Battle against Phishing: Dynamic Security Skins," *Proc. 2005 Symp. Usable Privacy and Security* (SOUPS 05), 2005, pp. 77–88.

Ari Juels is a professor at Cornell Tech. Contact him at juels@cornell.edu.

Thomas Ristenpart is an assistant professor in the University of Wisconsin's Department of Computer Sciences. Contact him at rist@cs.wisc.edu.

cn *Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.*

Have an idea for a future article?
Email editors Peter Gutmann (pgut001@cs.auckland.ac.nz), David Naccache (david.naccache@ens.fr), and Charles C. Palmer (ccpalmer@us.ibm.com).