Password-Based Authentication: Preventing Dictionary Attacks

Saikat Chakrabarti and Mukesh Singhal
University of Kentucky

Password-based authentication is susceptible to attack if used on insecure communication channels like the Internet. Researchers have engineered several protocols to prevent attacks, but we still need formal models to analyze and aid in the effective design of acceptable password protocols geared to prevent dictionary attacks.

Authentication provides a means of reliably identifying an entity. The most common verification technique is to check whether the claimant possesses information or characteristics that a genuine entity should possess. For example, we can authenticate a phone call by recognizing a person's voice and identify people we know by recognizing their appearance.

But the authentication process can get complicated when visual or auditory clues aren't available to help with identification—for example, when a print spooler tries to authenticate a printer over the network, or a computer tries to authenticate a human user logging in.

A computer can authenticate humans through

- biometric devices such as retinal scanners, fingerprint analyzers, and voice-recognition systems that authenticate *who the user is*;
- passwords that authenticate *what the user knows*;
- and smart cards and physical keys that authenticate *what the user has*.

Because they're cheap and convenient, passwords have become the most popular technique for authenticating users trying to access confidential data stored in computers. However, password-based authentication is vulnerable to several forms of attack.

People generally select short, easily memorized passwords to log in to a server without considering that password-based authentication methods are susceptible to attacks if used on insecure communication channels like the Internet. Meanwhile, complex passwords might get lost or stolen when users write them down, defeating the purpose of constructing secure password-based authentication schemes in the first place.

COMMON MECHANISMS AND SECURITY CONCERNS

Transmitting a password in plaintext from the user to the server is the simplest (and most insecure) method of password-based authentication. To validate a user password, the server compares it with a password (either in plaintext or an image of the password under a one-way function) stored in a file. However, this method lets an adversary passively eavesdrop on the communication channel to learn the password.

Challenge-response protocol

To secure against passive eavesdropping, researchers have developed *challenge-response protocols*.[1] To initiate a challenge-response protocol, Entity A sends a message containing A's identity to Entity B. Then B sends A a random number, called a *challenge*.

A uses the challenge and its password to perform some computation and sends the result, called a *response*, to B. Then B uses A's stored password to perform the same computation and verify the response. Since B chooses a different challenge for every run of the protocol, an

Published by the IEEE Computer Society

adversary can't simply eavesdrop, record messages, and resend them at a later time (a *replay attack*) to impersonate an entity.

Dictionary attacks

The challenge-response protocol is vulnerable to a *password-guessing attack*. In this kind of attack, we assume that an adversary has already built a database of possible passwords, called a dictionary. The adversary eavesdrops on the channel and records the transcript of a successful run of the protocol to learn the random challenge and response. Then the adversary selects passwords from the dictionary and tries to generate a response that matches the recorded one. If there's a match, the adversary has successfully guessed A's password.

After every failed matching attempt, the adversary picks a different password from the dictionary and repeats the process. This noninteractive form of attack is known as the *offline dictionary attack*.

Sometimes an adversary might try different user IDs and passwords to log in to a system. For popular Internet services like Yahoo!, the adversary can trivially choose any reasonable user ID due to the large number of registered users. An adversary can also find user IDs within interactive Web communities such as auction sites. If the system rejects the password as being incorrect for that particular user, the adversary picks a different password from the dictionary and repeats the process. This interactive form of attack is called the *online dictionary attack*.

Other security issues

Password-based authentication also can involve other security issues. Let's consider a scenario in which two entities, A and B, are trying to authenticate each other through a password protocol. An adversary can intercept messages between the entities and inject his own messages. In this *man-in-the-middle attack*, the adversary's goal is to play the role of A in the messages he sends to B and the role of B in the messages he sends to A.

In an *insider attack*, a legitimate user might try to attack other accounts in the system. Any additional information regarding a certain user might help in guessing that user's password.

PREVENTING OFFLINE DICTIONARY ATTACKS

Seeking convenience, people tend to choose weak passwords from a small sample space, which an adversary can easily enumerate. Thus, systems need something stronger than simple challenge-response protocols that can use these cryptographically weak passwords to securely authenticate entities. Such an authentication protocol would be deemed secure if, whenever an entity accepts an authentication session with another entity, it should have indeed participated in the authentication session.[2]

1. A : (E_A, D_A).
2. A → B : A, $K_{pwd}(E_A)$.
3. B : Compute $E_A = K^{-1}{}_{pwd}\,(K_{pwd}(E_A))$. Generate random secret key K_{AB}.
4. B → A : $K_{pwd}\,(E_A\,(K_{AB}))$.
5. A : $K_{AB} = D_A(K^{-1}{}_{pwd}(K_{pwd}(E_A(K_{AB}))))$. Generate unique challenge C_A .
6. A → B : $K_{AB}\,(C_A)$.
7. B : Compute $C_A = K^{-1}{}_{AB}(K_{AB}\,(C_A))$ and generate unique challenge C_B.
8. B → A : $K_{AB}\,(C_A, C_B)$.
9. A: Decrypt message sent by B to obtain C_A and C_B. Compare the former with his own challenge. If they match, go to next step, else abort.
10. A → B : $K_{AB}\,(C_B)$.
11. B : Decrypt message A sends and compare with challenge C_B. If they match, B knows that A has the ability to encrypt subsequent messages using key K_{AB}.

Figure 1. Algorithm 1: Encrypted key exchange. The EKE protocol uses a combination of symmetric and asymmetric cryptography.

Guarantees of mutual authentication are essential for remote users trying to access servers over insecure networks like the Internet. The goal of a password-based authentication protocol aimed at preventing offline dictionary attacks is to produce a cryptographically strong shared secret key, called the *session key*, after a successful run of the protocol. Both entities can use this session key to safely encrypt subsequent messages.

Encrypted key exchange

Steven Bellovin and Michael Merritt[3] made the first attempt to protect a password protocol against offline dictionary attacks. They developed a password-based encrypted key exchange (EKE) protocol using a combination of symmetric and asymmetric cryptography. Algorithm 1 in Figure 1 describes the EKE protocol, in which users A and B serve as the participating entities in a particular run of the protocol, resulting in a session key (stronger than the shared password) the users can later apply to encrypt sensitive data.

In Step 1, user A generates a public/private key pair (E_A, D_A) and also derives a secret key K_{pwd} from his password *pwd*. In Step 2, A encrypts his public key E_A with K_{pwd} and sends it to B. In Steps 3 and 4, B decrypts the message using the stored password of A, and uses E_A together with K_{pwd} to encrypt a session key K_{AB} and sends it to A.

In Steps 5 and 6, A uses this session key to encrypt a unique challenge C_A and sends the encrypted challenge to B. In Step 7, B decrypts the message to obtain the challenge and generates a unique challenge C_B.

1. A → B : A.
2. B → A : s.
3. A : $x = H(s, pwd)$; $K_A = g^a$.
4. A → B : K_A .
5. B : $K_B = v + g^b$.
6. B → A : K_B; r.
7. A : $S = (K_B - g^x)^{a+rx}$ and B : $S = (K_A v^r)b$.
8. A, B : $K_{AB} = H(S)$.
9. A → B : $C_A = H(K_A, K_B, K_{AB})$.
10. B verifies C_A and computes $C_B = H(K_A, C_A, K_{AB})$.
11. B → A : C_B.
12. A verifies C_B. Accept if verification passes; abort if not.

Figure 2. Algorithm 2: Secure remote-password protocol. SRP successfully eliminates plaintext equivalence.

In Step 8, B then encrypts both C_A and C_B with the session key K_{AB} and sends it to A. In Step 9, A decrypts this message to obtain C_A and C_B and compares the former with his own challenge. A match verifies the correctness of B's response.

In Step 10, A encrypts B's challenge C_B with the session key K_{AB} and sends it to B. In Step 11, B decrypts this message and compares it with his own challenge C_B. If they match, B knows that A can use K_{AB} to encrypt subsequent messages.

Bellovin and Merritt also developed augmented EKE (A-EKE),[4] which stores passwords under a one-way function. The objective is to prevent an adversary who obtains the one-way encrypted password file from mimicking the user to the host. They implemented A-EKE using digital signatures and a family of commutative one-way functions. Researchers subsequently developed a gamut of protocols that provide stronger security guarantees than EKE and have additional desirable properties.

The EKE protocol and its variants (except A-EKE) suffer from *plaintext equivalence*, which means the user and the host have access to the same secret password or hash of the password. Intuitively, there are disadvantages to plaintext equivalence.

Imagine a simple case in which entity A (the user) enters his password in the client software, which uses a one-way function to hash the password and sends the hashed password over the network to entity B (the server). An adversary can eavesdrop on the channel to obtain entity A's hashed password and can impersonate entity A by resending the hashed password later.

To understand the problem of plaintext equivalence, we can extend the simple case to more complex challenge-response protocols, like EKE. This vulnerability will arise whenever two entities share a secret and perform symmetric operations, however complex, based on the shared secret and exchanged messages.

Secure remote password

Thomas Wu[1] combined *zero-knowledge proofs* with *asymmetric key-exchange protocols* to develop *secure remote password* (SRP), a verifier-based protocol that eliminates plaintext equivalence. If the password is a private key with limited entropy, we can think of the corresponding verifier as a public key. It's easy to compute the verifier from the password, but deriving the password, given the verifier, is computationally infeasible.

However, unlike with a public key, the entity doing the validation can keep the verifier secret. All SRP computations are carried out on the finite field F_n, where n is a large prime. Let g be a generator of F_n. Let A be a user and B be a server. Before initiating the SRP protocol, A and B do the following:

- A and B agree on the underlying finite field.
- A picks a password pwd and a random salt s, and computes the verifier $v = gx$, where $x = H(s, pwd)$ is the long-term private key and H is a cryptographic hash function.
- B stores the verifier v and the salt s corresponding to A. Now, A and B can engage in the SRP protocol.

Alorithm 2 in Figure 2 describes the SRP protocol, which works as follows:

In Step 1, A sends its username A to server B. In Step 2, B looks up A's verifier v and salt s and sends A the salt. In Steps 3 and 4, A computes its long-term private-key $x = H(s, pwd)$, generates an ephemeral public key $KA = ga$ where a is randomly chosen from the interval $1 < a < n$ and sends K_A to B.

In Steps 5 and 6, B computes ephemeral public-key $K_B = v + gb$ where b is randomly chosen from the interval $1 < a < n$ and sends K_B and a random number r to A. In Step 7, A computes $S = (KB - gx)a + rx = gab + brx$ and B computes $S = (KAvr)b = gab + brx$. The values of S that A and B compute will match if the password A enters in Step 3 matches the one that A used to calculate the verifier v that is stored at B.

In Step 8, both A and B use a cryptographically strong hash function to compute a session key $K_{AB} = H(S)$. In Step 9, A computes $C_A = H(K_A, K_B, K_{AB})$ and sends it to B as evidence that it has the session key. C_A also serves as a challenge. In Step 10, B computes C_A itself and matches it with A's message. B also computes $C_B = H(K_A; C_A; K_{AB})$. In Step 11, B sends C_B to A as evidence that it has the same session key as A. In Step 12, A verifies C_B, accepts if the verification passes and aborts otherwise.

Unlike EKE, the SRP protocol doesn't encrypt messages. Since neither the user nor the server has access to the same secret password or hash of the password, SRP successfully eliminates plaintext equivalence. SRP is unique in its swapped-secret approach to developing a verifier-based, zero-knowledge protocol that resists offline dictionary attacks.[5]

A formal approach to prevention

Password protocols need more than heuristic arguments to provide security guarantees. The use of formal methods to analyze and validate security issues is of paramount importance in constructing "acceptable" password protocols.

Shai Halevi and Hugo Krawczyk[2] carried out the first rigorous security analysis of password-based authentication protocols, examining the use of password protocols for strong authentication and key exchange in asymmetric scenarios.

Password protocols need more than heuristic arguments to provide security guarantees.

In an asymmetric scenario, the authentication server can store a private key for public-key encryption, but the client uses a weak password and doesn't have a means to authenticate the server's public key via a trusted third party. Halevi and Krawczyk presented and analyzed the security of simple and intuitive password-based authentication protocols—like a generic encrypted challenge-response protocol and a mutual authentication/key exchange protocol. They also proved that every authentication protocol that attempts to resist offline dictionary attacks needs public-key encryption and demonstrated that they could build a secure key-exchange protocol, given any such password protocol.

Standard model

Other researchers including Mihir Bellare and his colleagues[6] subsequently proposed formal models for password-authenticated key exchange. However, the formal validations of security don't constitute proofs in the standard model. For example, Bellare used ideal ciphers to achieve provable security. The standard model is commonly used in modern cryptography.

Since we still don't have proofs that any of the standard cryptographic building blocks have computational lower bounds, achieving common cryptographic goals requires making some complexity-theoretic hardness assumptions.[7] Examples of such assumptions include the following:

- Factoring the product of large primes is hard.
- Computing the discrete logarithm is hard in certain sufficiently large groups.
- The Advanced Encryption Standard (AES) is a good pseudorandom permutation.

Although the proofs performed under the standard model use such assumptions, the cryptographic community widely accepts the standard model.

Alternative models

When constructing proofs, researchers often resort to an alternative when proofs in the standard model are unappealing or provably impossible (http://eprint.iacr.org/2005/210.pdf). One such model is the *random-oracle model*, which constitutes a public random function that takes any string $s \in \{0,1\}^*$ as input and outputs n bits. For every input string, the output is uniform and independent of all other outputs. Powerful as it is, the random oracle doesn't exist in the real world. A cryptographic hash function usually instantiates it.

Another alternative, the *ideal-cipher model,* uses a *block cipher*, an algorithm that accepts a fixed-length block of plaintext and a fixed-length key as input and outputs a block of cipher text that's the same length as the block of plaintext. The block cipher is constructed with a k-bit key and an n-bit block size and is chosen uniformly from the set of all possible block ciphers of the same form. Somewhat analogous to the random oracle model, a practical block cipher must instantiate the ideal-cipher model's black box.

If a password-authenticated key-exchange protocol uses the random-oracle model or the ideal-cipher model to construct a formal analysis of its security and achieves provable security under that model, what guarantees do we get once we instantiate those alternative models?

Some cryptographers have doubted protocols using such alternative models to claim provable security. There are cases where instantiations of idealized models have resulted in erroneous outcomes. Zhu Zhao and his colleagues[7] presented examples of real ciphers that resulted in broken instantiations of Bellare and his colleagues' password protocol.

To the best of our knowledge, achieving provable security in a password-based authentication protocol (preventing offline dictionary attacks) based on the standard model is still an open problem. At the current stage of research, the best we can do is aim for achieving provable security under a formal model, maybe an idealized one, and not construct a protocol claiming security attributes based on heuristic arguments.

PREVENTING ONLINE DICTIONARY ATTACKS

Password-based authentication will continue to be the most commonly used authentication technique, and hacking and identity thefts will be the wave of the future. However, several techniques are available to help withstand online dictionary attacks, where the adversary tries to impersonate a user to the server by repeatedly trying different passwords from a dictionary of passwords.

Prevention techniques and drawbacks

In 2002, online dictionary attacks were blamed for eBay accounts being taken over and used to set up

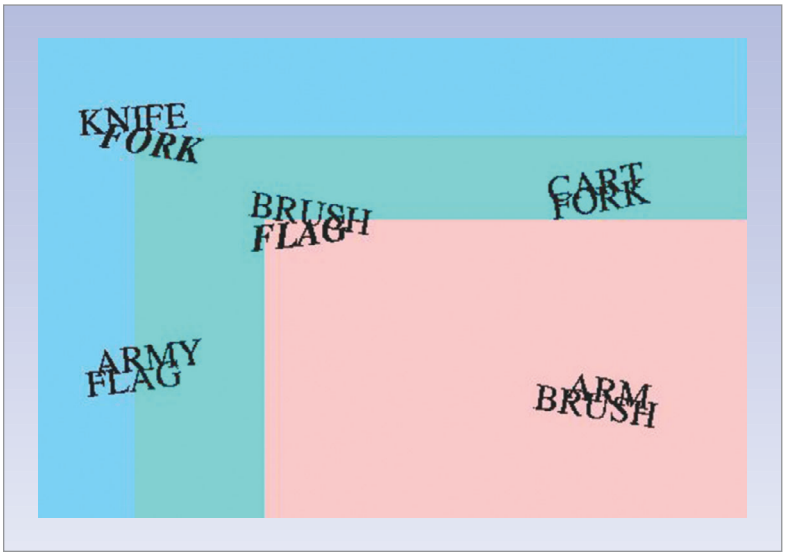

Figure 3. A Gimpy is a Captcha based on optical-character recognition.

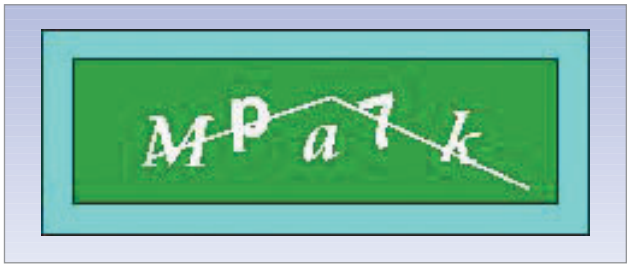

Figure 4. Yahoo! uses EZ-Gimpy, which presents a distorted image over a textured background.

fraudulent auctions (http://news.zdnet.com/2100-9595_22-868306.html). Users clearly found this vulnerability unacceptable. The attacks pointed to the need for countermeasures, and Benny Pinkas and Tomas Sander responded with several mechanisms.[8]

Delayed response. After receiving a user ID/password pair, the server sends a slightly delayed response. This prevents an adversary from checking a sufficiently large number of passwords for a user ID in a reasonable amount of time.

However, just as a server can process several user logins in parallel, an adversary can try several login attempts in parallel to work around the delayed-response approach. For popular Internet services like Yahoo!, the adversary can trivially choose a user ID due to the large number of users. User IDs also can be found in interactive Web communities like auction sites.

Account locking. To prevent an adversary from trying many passwords for a particular user ID, systems can lock accounts after a certain number of unsuccessful login attempts. However, an adversary can mount a denial-of-service attack by choosing a valid user ID and trying several passwords until the account gets locked. This would cause a great inconvenience to the owners of locked accounts, and setting up customer service to handle user calls regarding locked accounts wouldn't be cost-effective.

Performing extra computation. Originally developed to combat junk e-mail, this technique requires a user to perform some nontrivial computation and send proof of it while trying to log in.[9] The idea is that the computation would be negligible for a single login attempt, but too expensive for a large number of login attempts.

For example, a server could require the following computation to be performed for every login attempt: Choose a value x so that the last 20 bits of $H(x$, user ID, password, time) are all 0, where H is a cryptographic hash function like SHA. If we assume H to be preimage resistant (given a message digest y, it's computationally infeasible to find x, such that $y = H(x)$), it would be necessary to check 219 values for x on the average to satisfy the condition.

A legitimate user might do this computation once, presenting a negligible overhead. However, performing this computation repeatedly for a large number of trial login attempts would present an extreme burden. The user's computer must run special software for the computation. In addition, the adversary might have a more powerful computer, and since the computation shouldn't be too time-consuming for a legitimate user, the adversary might have an edge in performing the dictionary attack.

Pinkas and Sander[8] observed that an automated program must carry out such interactive forms of attacks, whereas legitimate users are humans. Thus, any login attempt must involve a test that a person can easily pass but an automated program can't.

Reverse Turing tests

Colorful images with distorted text have become commonplace at Web sites like Yahoo!, Hotmail, and PayPal. They're called reverse Turing tests (RTT) or Completely Automated Public Turing Tests to Tell Computers and Humans Apart (Captcha; http://captcha.net). Humans can easily pass the tests, but computer programs can't, even if they're knowledgeable about complete descriptions of the algorithms that created such tests.

Pinkas and his colleagues have implemented RTTs to prevent dictionary attacks. People can easily use the login accompanied by the RTT, but automated programs trying to carry out an online dictionary attack can't. The RTT should constitute a test with a small probability of a random guess producing a correct answer. For example, a test asking the user to identify whether an image is a man or a woman wouldn't be permissible since a random guess produces a correct answer with 50 percent probability.

Figure 3 shows Gimpy, a Captcha based on optical-character recognition. It renders a distorted image containing 10 words (some repeated), overlaid in pairs. Human users can easily read three different words from the distorted image, but computer programs can't.

Yahoo! uses an easier version called EZ-Gimpy, which presents a distorted image of a single word presented on a cluttered textured background, as Figure 4 shows.

Figure 5 illustrates Bongo, a Captcha that presents a visual pattern-recognition problem. Bongo asks users to distinguish between two blocks, then presents a single block and asks the user to determine whether it belongs to the right or left block.

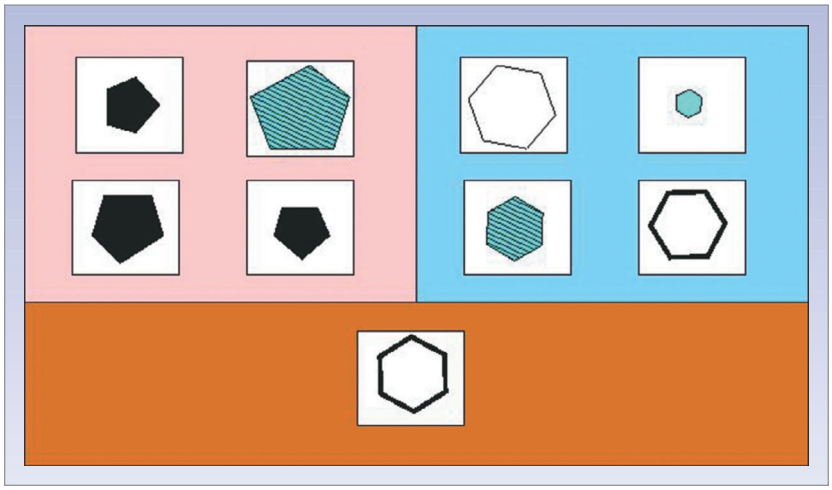

Figure 5. Bongo. This Captcha asks users to solve a visual pattern-recognition problem.

A basic password-based authentication protocol using RTTs requires the user to pass an RTT before entering the user ID and password. This method has some drawbacks because it's demanding to ask users to solve an RTT for every login attempt. Currently, RTTs are more commonly generated for filling out online registration forms.

It's unknown whether the algorithm-generating RTTs can scale up to be used for every login attempt. Pinkas assumed that users log in from a limited set of computers containing activated cookies. So instead of using RTTs for every login, the user is asked to pass an RTT when initially trying to log in from a new computer or when entering a wrong password. The decision whether to present an RTT or not is a deterministic function of the entered user ID/password pair.

Stuart Stubblebine and Paul van Oorschot[10] observed that RTT-based protocols are vulnerable to *RTT relay attacks*. Suppose an adversary wants to perform an online dictionary attack at the eBay Web site. For this, it needs correct responses to the RTTs. But an adversary can hack a high-volume Web site such as cnn.com and install attack software, which initiates a fraudulent attempt to login at ebay.com when a visitor goes to cnn.com. The RTT challenge is redirected to the user trying to view the cnn.com page.

Many nontechnical users will solve the RTT, unaware that the attack software will relay the answer to eBay, thus solving the RTT challenge. Solving the RTT, along with a sufficient number of password guesses, can crack an eBay account password. To counter these kinds of RTT relay attacks, Stubblebine developed a protocol based on a user's login history, suggesting modifications to Pinkas's RTT-based protocol.

Stubblebine suggested that only trustworthy machines store cookies. He also recommended that systems track users' failed-login attempts and set failed-login thresholds. His analysis of the protocol showed that it's more secure and user friendly.

Engineers or security architects wanting to select appropriate authentication techniques should be careful if they want to implement Captchas commercially. Hewlett-Packard holds a US patent on several forms of Captchas[11] and Yahoo! has applied for a patent on an image-verification system to prevent messaging abuse.[12]

Password-based authentication should continue to be the most common technique for user verification, as will attacks on it through a combination of hacking and identity theft. Password protocols preventing offline dictionary attacks need more than heuristic arguments to provide a guarantee of security. Although researchers have developed formal models for password-authenticated key exchange, the formal validations of security don't constitute proof in the standard model. While RTTs serve as tests that humans, but not automated programs, can pass, it's demanding to ask users to solve an RTT for every login attempt. Consequently, effective design of password protocols using RTTs requires a good balance between tight security and user friendliness.

Acknowledgments

The authors thank the anonymous reviewers whose valuable comments helped improve this article. This research was partially supported by grant no. T0505060 from the US Treasury Department.

References

1. T. Wu, "The Secure Remote Password Protocol," *Proc. Network and Distributed System Security* (NDSS), The Internet Soc., 1998, pp. 97-111; http://isoc.org/isoc/conferences/ndss/98/wu.pdf.

2. S. Halevi and H. Krawczyk, "Public-Key Cryptography and Password Protocols," *ACM Trans. Information System Security*, ACM Press, vol. 2, no. 3, 1999, pp. 230-268.
3. S.M. Bellovin and M. Merritt, "Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks," *Proc. IEEE Symp. Security and Privacy*, IEEE CS Press, 1992, pp. 72-84.
4. S.M. Bellovin and M. Merritt, "Augmented Encrypted Key Exchange: A Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise," *Proc. ACM Conf. Computer and Comm. Security*, ACM Press, 1993, pp. 244-250.
5. D. Bleichenbacher, "Breaking a Cryptographic Protocol with Pseudoprimes," *Proc. 8th Int'l Workshop Theory and Practice in Public-Key Cryptography* (PKC 2005), LNCS 3386, Springer, 2005, pp. 9-15.
6. M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated Key Exchange Secure Against Dictionary Attacks," *Advances in Cryptology—EUROCRYPT 2000, Proc. Int'l Conf. Theory and Application Cryptographic Techniques,* LNCS 1807, Springer, 2000, pp. 139-155.
7. Z. Zhao, Z. Dong, and Y. Wang, "Security Analysis of a Password-Based Authentication Protocol Proposed to IEEE 1363," *Theoretical Computer Science*, vol. 352, no. 1, Elsevier, 2006, pp. 280-287; http://sis.uncc.edu/~yonwang/papers/TCSsrp5. pdf.
8. B. Pinkas and T. Sander, "Securing Passwords Against Dictionary Attacks," *Proc. 9th ACM Conf. Computer and Comm. Security*, ACM Press, 2002, pp. 161-170.
9. C. Dwork and M. Naor, "Pricing via Processing or Combating Junk Mail," E.F. Brickell, ed., *Advances in Cryptology—CRYPTO '92*, LNCS 740, Springer, 1993, pp. 139-147.
10. S.G. Stubblebine and P.C. van Oorschot, "Addressing Online Dictionary Attacks with Login Histories and Humans-in-the-Loop," *Financial Cryptography*, LNCS 3110, Springer, 2004, pp. 39-53; www.ccsl.carleton.ca/paper-archive/pvanoorschot-fc-04.pdf.
11. M.D. Lillibridge et al., *Method for Selectively Restricting Access to Computer Systems*, US patent 6,195,698, Patent and Trademark Office, 1998.
12. *Method and System for Image Verification to Prevent Messaging Abuse*, US patent application, 2004/0199597, Patent and Trademark Office, 2004.

Saikat Chakrabarti *is a PhD student in the Computer Science Department at the University of Kentucky. His research interests are network and distributed-system security and applied cryptography. He received a BS in electrical engineering from Bengal Engineering and Science University, Shibpur, India. He is a student member of the IEEE and the ACM. Contact him at schak2@cs.uky.edu.*

Mukesh Singhal *is the Gartner Group Endowed Chair in Networking in the Computer Science Department at the University of Kentucky. His research interests are computer network security, distributed computing and operating systems, and wireless and high-speed networks. He received a PhD in computer science from the University of Maryland. He is an IEEE Fellow. Contact him at singhal@cs.uky.edu.*