Computers & Security (2005) ■, ■–■

Computers & Security

www.elsevier.com/locate/cose

A new protocol to counter online dictionary attacks

Vipul Goyal [a,*], Virendra Kumar [a], Mayank Singh [a], Ajith Abraham [b], Sugata Sanyal [c]

[a] *Crypto Group, Institute of Technology, Banaras Hindu University, India*
[b] *IITA Professorship Program, School of Computer Science and Engineering, Chung-Ang University, South Korea*
[c] *School of Technology and Computer Science, Tata Institute of Fundamental Research, India*

Revised 25 August 2005; accepted 21 September 2005

KEYWORDS
Authentication;
Online dictionary attacks;
Hash functions;
Cryptographic protocol;
Passwords

Abstract The most popular method of authenticating users is through passwords. Though passwords are the most convenient means of authentication, they bring along themselves the threat of dictionary attacks. While offline dictionary attacks are possible only if the adversary is able to collect data for a successful protocol execution by eavesdropping on the communication channel and can be successfully countered by using public key cryptography, online dictionary attacks can be performed by anyone and there is no satisfactory solution to counter them. In this paper, we propose an authentication protocol which is easy to implement without any infrastructural changes and yet prevents online dictionary attacks. Our protocol uses only one way hash functions and eliminates online dictionary attacks by implementing a challenge–response system. This challenge–response system is designed in a fashion that it hardly poses any difficulty to a genuine user but is extremely burdensome, time consuming and computationally intensive for an adversary trying to launch as many as hundreds of thousands of authentication requests as in case of an online dictionary attack. The protocol is perfectly stateless and thus less vulnerable to denial of service (DoS) attacks.
© 2005 Elsevier Ltd. All rights reserved.

* Corresponding author.
E-mail addresses: vipul.goyal@cse04.itbhu.org (V. Goyal), virendra.kumar@eee06.itbhu.org (V. Kumar), mayank.singh@eee06.itbhu.org (M. Singh), ajith.abraham@ieee.org (A. Abraham), sanyal@tifr.res.in (S. Sanyal).

0167-4048/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2005.09.003

Introduction

Currently, a vast majority of systems use passwords as the means of authentication. Passwords are very convenient for the users, easier to implement and consequently very popular too. Although more secure authentication schemes have been suggested in the past, e.g., using smartcards, none of them have been in widespread use in the consumer market. The password based authentication, although very convenient, has some drawbacks due to the very nature of this system. As is obvious, humans have great tendency to choose relatively short and simple passwords that they can remember. Thus the chosen passwords belong to a small and predictable domain. They are in fact very much susceptible to exhaustive search or dictionary attacks (Klein, 1990; Morris and Thompson, 1979). There are several instances of such attacks on various systems throughout the world (Hackers find new way to bilk eBay users, 2002).

Password based systems mainly suffer from offline and online dictionary attacks. In an offline dictionary attack the adversary eavesdrops on the communication channel to record data for a successful protocol execution. The adversary then goes offline and tests passwords against the recorded protocol execution data without contacting the server at all. In an online dictionary attack, the adversary tries the possible passwords by logging in online. Offline dictionary attacks, although severe, can be prevented by various protocols using public key cryptography, suggested in the past. The first password based authentication protocol, secure against offline dictionary attacks is called EKE and was designed by Bellovin and Merritt (1992). Since then a number of excellent protocols addressing this problem have been proposed. An interested reader may refer to Halevi and Krawczyk (1999). However, no satisfactory measures to curb online dictionary attacks have been suggested so far. There are some methods to deal with such attacks but some of them have security flaws and the others are impractical in terms of usage. In this paper we will discuss some of the existing protocols, their strengths and weaknesses and suggest a protocol to deal with the problem discussed.

Our protocol employs fast one way hash functions (ANSI, 1993) and reduces the number of possible guesses in a given time period. This is done by asking the client to compute the response for a given challenge (Goyal et al., 2005). The computation of this response is designed to be a time consuming operation. Special care is taken to ensure that the client is not able to reuse the computation and make the protocol perfectly stateless.

The rest of the paper is organized as follows: In section ''Related research'' we discuss the existing protocols — their strengths, weaknesses and flaws (if any). Sections ''Basic idea of the protocol'' and ''The protocol'' are dedicated to the protocol we have proposed where we discuss the basic idea behind the design of the protocol and then we discuss the details of the protocol. In section ''Improvements and modifications'' we discuss a few improvements and changes for use in specific situations. Finally we conclude the paper in the last section.

Related research

Password based systems are vulnerable to online dictionary attacks. These attacks are difficult to curb and hence they pose a major problem in the functioning of password based systems. Counter measures adopted to prevent the online dictionary attacks are much more expensive and yet ineffective. In this section we mention a few measures adopted these days to prevent this attack and discuss the drawbacks which they suffer from.

Account locking

After a few fixed number of unsuccessful login attempts, the account of the particular user is locked for sometime. No doubt this is helpful in preventing the possible dictionary attack by limiting the number of wrong guesses in a given time, but as we will discover in what follows, there are various problems in such a system in a large network.

If account locking is adopted then the system will become susceptible to denial of service (DoS) attack in which an adversary can knowingly launch many login requests so that the user's account may get locked for a fixed period. Thus, the genuine users are deprived of the service in that period. Yahoo!, for example, reports that users who compete in auctions use these methods to block the account of other users competing in the same auctions. This attack may be worrisome to mission critical applications, for example to enterprises whose employees and customers use the web to login to their accounts. In a similar manner, distributed denial of service (DDoS) attack may also be launched on a system employing the account locking feature. In this, the attacker could plant hidden agents around the web and all the agents could start operating at a specific time. Thus, they could block virtually a large proportion of the accounts of the attacked server by trying to login into accounts in that server using random passwords.

Another major drawback of the ''account locking feature'' is that since it causes user accounts to be locked, either by mistake (e.g. by users who do not type their passwords correctly) or as a result of dictionary attacks, the service provider must operate customer service centres to handle calls from users whose accounts are locked. The cost of running these centres is high, and is estimated to cost more than $25 per customer call. Imagine that each user locks his account once in every five years, then the service cost, per user, per year, is at least $5. A news article (Hackers find new way to bilk eBay users, 2002) suggested that eBay had not implemented account locking features due to the costs of operating customer support centres.

An option here for the service provider could be to automatically unlock the account after a fixed amount of time (e.g. 12 h). But then, it is easy for anyone to keep the account of a customer always locked (e.g. by using programs which send login requests with random passwords after every 12 h) and thus totally depriving the customer of the service.

Despite the above serious problem, account locking is still a commonly adopted countermeasure against online dictionary attacks. Major web based service providers like Yahoo! use this approach to counter online dictionary attacks.

Delayed response

In this scheme, the server provides a delayed response to the user request, say for example, not faster than one answer per second. This may prevent an attacker from checking enough number of passwords in a reasonable time.

This scheme is very effective for local machines in which the user has to login to the computer using a physically attached keyboard. However, it is totally ineffective in a network environment. The attacker can try many login attempts in parallel and circumvent the timing measure using the fact that user logins are typically handled by servers that can handle many login sessions in parallel. For example, the attacker can send a login attempt every 10 ms, thus obtaining a throughput of 100 login attempts per second, regardless of how long the server delays the answer to a given login attempt. This scheme also suffers from global password attacks. A system that has many user accounts and enables logins over a network accessible to the adversary, suffers from such attacks. In many situations, an attacker is interested in breaking any account in the system, rather than targeting a specific account.

Use of CAPTCHA

CAPTCHA stands for Completely Automated Public Turing Test to Tell Computers and Humans Apart (Pinkas and Sander, 2002). In this scheme, some challenge is put forward to the user while attempting to login. It has been established that these challenges, for example a distorted and cluttered image of a word with textured background, are easy for humans to respond but rather difficult for computers (an online attacker is essentially a programmed computer) to answer. Until recently, this scheme was an effective countermeasure against online dictionary attacks. However, due to recent developments in Artificial Intelligence and Computer Vision, programs are available which can quickly interpret and answer these challenges. EZ-Gimpy and Gimpy for example are word based CAPTCHAs that have been broken by Greg Mori and Jitendra Malik of UC Berkeley Computer Vision Group (Berkeley, 2004). Due to these developments, even CAPTCHA is not considered to be a secure technique to prevent online dictionary attacks.

A few major web based service providers who were earlier using the CAPTCHA technique have now resorted to highly inconvenient account locking in order to counter online dictionary attacks. Clearly, a better and elegant method for solving this pressing problem is required.

Basic idea of the protocol

Looking at the various instances of online dictionary attacks, one point is very clear. All these are possible because they pose very little difficulty to the attacker. Due to the particular structure of the present systems, it is immaterial whether one launches one request or one million requests — the response time is same in both the cases. This is a major advantage for the attackers. Attackers use computers to launch thousands of requests in a fraction of a second and are thus able to achieve their purpose of testing a large number of passwords for validity. It may thus be concluded that the dictionary attacks can be prevented if the whole process of bulk requesting is made complicated, time consuming and costly without hampering the ease of use for genuine users.

The proposed protocol has a major advantage that it can completely eliminate the possibility of a large number of password guesses in a small time interval by making it difficult and costly. Even the dedicated computers used by attackers will face much difficulty in launching several requests at the same time. Along with the prevention of online dictionary attacks, the proposed protocol is stateless and thus less vulnerable to DoS attacks (Rivest, 1998).

Note that our protocol does not use public key cryptography. This means that the protocol is vulnerable to offline dictionary attacks if an adversary records data for a successful protocol execution by eavesdropping on the communication channel.[1] In order to resist offline dictionary attacks, the server and client may first establish an SSL connection and the session key could be used to encrypt different messages of the protocol. Major web based service providers like Yahoo! and Hotmail, etc already use SSL for protecting login data in transit. Hence, this does not require any infrastructural changes. In cases where performance degradation due to public key cryptography is a concern, we provide a variant of our protocol which makes it very difficult to launch offline dictionary attacks.

Our protocol uses only fast one way hash functions (ANSI, 1993). The user and the server are required to perform a few hash calculations for logging in. The system is deliberately made time consuming and computationally intensive for the client to ensure that it is not able to make a large number of authentication requests per second. However, our system is extremely efficient for the server. Hash functions, unlike public key cryptography, can be computed using minicomputers, handhelds or by easily available mobile code like Java or JavaScript Program.

The protocol

This is a four pass protocol and only hash computations are employed throughout the protocol. Two out of four messages are simple message exchange without any encryption. The remaining two messages involve hash computation: once by the user and once by the server, as illustrated in Fig. 1. The protocol presents a challenge for the user by the server and the user can login only after cracking the presented challenge which requires some computation time. This computation time can be easily increased or decreased by the server at will. We describe our protocol message by message touching each and every point. The protocol description is followed by a brief discussion of the different security measures taken to prevent the major threats. Throughout the discussion the user is assumed to be Alice (A) and the server to be Bob (B).

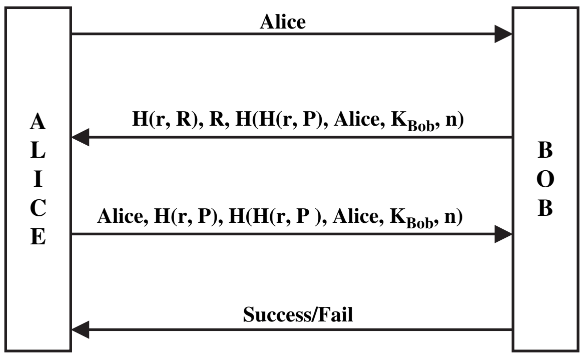
Alice
H(r, R), R, H(H(r, P), Alice, K_Bob, n)
Alice, H(r, P), H(H(r, P), Alice, K_Bob, n)
Success/Fail
ALICE
BOB

Figure 1 Different passes of the protocol. The following notations have been used: K_{Bob} = secret key of the server Bob, known only to him and no one else; P = password of the user; n = number of unsuccessful login attempts, to be stored by the server; r = a 20-bit random number; R = a 128-bit random number; MAC = message authentication code, to be sent by the server to the client; $H(X)$ = hash value of X, using standard hash function.

Protocol description

Message 1. A → B: Alice

This is a simple request by the user Alice that she wants to login.

Message 2. B → A: $H(r, R)$, R, $H(H(r, P)$, Alice, K_{Bob}, $n)$

In response to the request sent by the user, the server sends the message 2 in which it sends a challenge $H(r, R)$, the value of R and the message authentication code (MAC) $H(H(r, P)$, Alice, K_{Bob}, $n)$. The challenge is the hash of two random numbers r (20-bit) and R (128-bit) appended together. The user has to find out r from the given hash value and the value of R. r may be any possible 20-bit number. We will discover shortly the various purposes for which r and R have been used. The third part of the message, MAC is again a hash value and unintelligible to anyone other than the server. This hash can be regenerated only by the server as the secret key K_{Bob} is known only to the server. Note that the client does not use this MAC in anyway. It only has to return the supplied MAC to the server in the next step so that the server does not have to store it. This MAC is used by the server to check

[1] Halevi and Krawczyk (1999) gave a mathematical proof suggesting that no symmetric key password based authentication protocol can resist offline dictionary attacks and it is mandatory to use public key cryptography to design such protocols.

the correctness of the value of r found by the user and also for the freshness of the message when the user replies with the message 3 as we will see later.

To find out the value of r, the user has to check the hash values of all the possible 20-bit numbers appended with the value of R. This computation is bit intensive and may require considerable time (about 5 s or even more depending on the system used). If instead of two random numbers only one large random number is used, then this computation time is very large and hence the user will be over burdened, which is undesirable. And if only a small 20-bit random number is used, then the attacker might store the hash values of all the possible 20-bit random numbers and could easily bypass the computation involved by simply searching for the correct value of r from the corresponding stored hash values. The use of two random numbers, one of 20 bits and the other of 128 bits thus fulfils two purposes. First, it gives just the right amount of computation to the user so that the online dictionary attacks are effectively countered as well as a genuine user is not disturbed. Secondly, it prevents the possibility of pre-computation of hash values for all possible 20-bit numbers. Thus, the number R effectively acts as a salt (Salt (cryptography); Ritter) in the computation of the number r.

The user, after receiving the second message, does the required calculations and finds the value of r, and then it proceeds with the third message.

Message 3. A → B: Alice, $H(r, P)$, MAC

In order to make the protocol stateless, this step has been made independent of the previous steps, i.e. the client initiates the connection again after doing the required computation and starts with the 3rd step of the protocol directly.

The user, after receiving the second message, finds the value of r from the given values and then sends her name, hash of the found value of r appended with the password P, and the MAC. In the message, the values of r and P could have been sent directly (without hashing) but it has been hashed to make the protocol secure against an eavesdropper.

The server, after receiving this message, finds out the hash of the sent $H(r, P)$ appended with the id of Alice, his secret key (K_{Bob}) and the stored value of n. It then compares the obtained hash value with the sent MAC. If they match, the login attempt is successful, else the login attempt fails and the server increments the value of n.

The use of the MAC is that it authorizes the supplied r to be the response of a challenge generated by the server and prevents the freshness attack, in which the attacker may use the same set of values again and again. We have used the value of n (number of unsuccessful attempts) in the computation of the MAC which is dynamic. So, a repeated use of message 2 is not possible as n increments on every unsuccessful attempt.

Here, an important point to observe is that n does not increment on a successful attempt. This is an interesting feature making the protocol friendly to the legitimate users. This means that if the user was successful in her last login attempt, she would be allowed to bypass the computation involved by reusing the last computation. Thus a legitimate user may actually be required to perform the computation only for the first time she tries to login. For every subsequent login attempts, the last computation could be reused as long as the login attempt does not fail.

By the use of MAC, the server is also relieved from the burden of storing the current values of r, R for checking the correctness of the value sent by the client. This makes the protocol perfectly stateless.

Message 4. B → A: success/fail

This is a simple reply by the server that the information provided by the user was correct or not. If found correct, the login attempt is successful, otherwise the user has to start all over again with the first message.

The protocol presented above is designed to eliminate the online dictionary attacks. For every login attempt, the user has to compute the value of r which is supplied by the server as a challenge. This computation requires some time which may vary from computer to computer. This computation time can be adjusted by simply varying the size of number r to keep pace with the computational speed of a processor which increases progressively with technological improvement. This required computation time is to discourage the online dictionary attack in which a dedicated computer launches thousands of login requests in a few seconds. By incorporating this protocol, the number of authentication requests possible in a given period is reduced drastically and hence the whole process of launching online dictionary attacks becomes very difficult and costly.

A brief discussion

In order to better understand the protocol, we discuss the different instances where an adversary may try to defeat the scheme and how the protocol resists such attempts.

In our scheme, the server is not supposed to store the value of either r or R and hence it verifies the values supplied by the user (in message 3) only from the supplied MAC. An attacker might be tempted to use the same MAC and hence reuse the computation for different login attempts. However, such an attempt is countered by our protocol. The server uses the stored n to compute the MAC. Since the value of stored n would be more than the value of n in the sent MAC, both the MACs would not match. Hence, the attempt to reuse the computation fails. An attacker, under no circumstances, will be able to change the MAC for different set of values of $H(r, P)$ and n since the secret key used in the MAC is not known to any body except the server.

In a similar way, an attempt to use the same values of message 3 (computed for one user id) for different user ids will easily be countered by the server since the user id is also used in the MAC computation.

Improvements and modifications

The presented protocol does not take care of the situation in which the server itself can be compromised. This is because the server is required to store the user password in plaintext since it was used for the computation of $H(r, P)$ in the computation of MAC. The protocol can be augmented so that the server only stores a one way hash $H(P)$ of the password and the authenticating user is required to know the actual password itself. There are two such variations possible.

(a) The first one is simple yet effective in maintaining the desired secrecy of the password. In this, the server stores the hash of the password and the user is required to send the password in plaintext. Hence, even if the server is compromised, the attacker is not able to find out the actual password, thus preventing the attacker from logging in successfully on behalf of the user. Use of SSL session is strongly recommended in this variant of the protocol, as otherwise the password would be directly obtained from message 3. The different passes of the protocol after this modification are as follows. Note that the MAC has also changed.
A $\rightarrow$ B: Alice
B $\rightarrow$ A: $H(r, R)$, R, MAC
A $\rightarrow$ B: Alice, r, P, MAC
B $\rightarrow$ A: success/fail
MAC $= H(r, H(P)$, Alice, K_{Bob}, $n)$

(b) The second variant is relatively complex but does not make it mandatory to use SSL protection. It employs the concept of lamport hashes (Lamport, 1981). To begin with, the server stores $H^m(P)$ (which is the mth hash of P) and the user is required to supply $H^{(m-1)}(P)$ as a password. Once the user has successfully logged in, the stored $H^m(P)$ is replaced by the supplied $H^{(m-1)}(P)$. Thus, next time the user would be required to supply $H^{(m-2)}(P)$. This process continues simply for m successful login attempts. Although it may seem that the user is required to re-initialize the system by choosing a different password after m successful logins, there are recently designed efficient techniques (Goyal, 2004a,b) which allow infinite number of login in the lamport system. Messages for the ith execution of the protocol are given below.
A $\rightarrow$ B: Alice
B $\rightarrow$ A: $H(r, R)$, R, MAC
A $\rightarrow$ B: Alice, r, $H^{(i-1)}(P)$, MAC
B $\rightarrow$ A: success/fail
MAC $= H(r, H^i(P)$, Alice, K_{Bob}, $n)$

Thus, we see that we have augmented the protocol so that the server is not required to store plaintext password and use of SSL therefore is not mandatory.

(c) As discussed earlier, it may be desirable to resist offline dictionary attacks without using SSL or other public key cryptographic techniques due to efficiency concern. Although this is theoretically impossible (Halevi and Krawczyk, 1999), we design a variant which makes it very difficult to launch successful offline dictionary attacks. A minor variation in the messages produces interesting results. In message 2, if $H(r, R)$ is changed to $H(r, P, R)$ (with other things unchanged), then the protocol is highly effective in preventing offline dictionary attacks.
The different passes of the protocol are as follows:
A $\rightarrow$ B: Alice
B $\rightarrow$ A: $H(r, P, R)$, R, MAC
A $\rightarrow$ B: Alice, $H(r, P)$, MAC
B $\rightarrow$ A: success/fail
MAC $= H(H(r, P)$, Alice, K_{Bob}, $n)$

Now, in such a situation let us try to analyze the system by assuming that the SSL session key has not been used to protect the different passes of our protocol. The time required by a legitimate user to find out r from the values supplied in message 2 is not affected, as the values of P and R used in the hash $H(r, P, R)$ are known to the user. The only thing unknown to the user is r. So the required computation is similar to that in original protocol. Taking standard values of time required in hash computation to be $t = 0.005$ ms, the maximum time required by the user to compute r will be $2^{20} \times t$ i.e. $2^{20} \times 0.005 \times 0.001$ ($=5.24288$ s) which is 5 s approximately.

Let n be the average number of guesses in an offline dictionary attack before the actual password is found out. Now, since the attacker does not know the value of P as well as r, to find the correct values of r and P from message 2 he will require $2^{20} \times n$ hash computations (i.e. he will have to try all possible combinations of passwords and 20-bit digits). Taking standard value of n to be 10 millions, the time required will be $2^{20} \times 10{,}000{,}000 \times 0.005 \times 0.001$ s $= 52{,}428{,}800$ s $= 1.6625$ years. Note that this may actually be much more for a reasonably good password.

To see how difficult it is to successfully launch an offline dictionary attack without this technique, i.e. in the original protocol proposed in section The protocol, we compute the time taken to launch offline dictionary attack in the original protocol under the same setting. The time taken by the attacker to compute 'r' from message 2 will be equal to $(2^{20}) \times t$ (i.e. he will try all the possible 20-bit digits) and then the time taken to compute the password from message 3 will be $n \times t$. So, the total time required to find out the password from the two messages is $(2^{20} \times t + n \times t)$. Evaluating this expression, we get the time to be $(5.24288 + 50) = 55.24288$ s.

Thus, it is clear that this variant is very effective in the prevention of offline dictionary attacks. This variant should be used when the protocol execution is not protected by SSL due to performance concerns.

Conclusions

There are no satisfactory means to counter online dictionary attacks. In this paper, we addressed the problem of online dictionary attacks and presented an authentication protocol to counter the same. The proposed protocol uses only fast one way hash functions and is based on a challenge—response system. Before logging in, the client is required to compute the response to the presented challenge. Computing this response is deliberately designed to be a moderately time consuming operation, thus ensuring that the client is not able to launch a large number of login requests in a small amount of time. The protocol is designed in a fashion such that the computation of this response hardly poses any problems for a legitimate user (who may reuse the last computation), but is extremely time consuming and costly for an adversary trying to launch thousands of login requests per second. Finally, we provided three variants of our protocol. The first two are concerned with augmenting the protocol so that the server is not required to store the password in plaintext. The third one is concerned with removing offline dictionary attacks in case public key cryptographic protection is not used.

Future work involves modifying the protocol such that the size of r and hence the required computation increases dynamically as the server encounters a large number of unsuccessful attempts in a small time window. It is to be noted that the presented technique could be used to address the problem of eliminating more general denial of service attacks on web servers by limiting the number of requests per second in a similar fashion.

References

ANSI X9.30 (PART 2). American National Standard For Financial Services — public key cryptology using irreversible algorithms for the financial services industry — part 2: the secure hash algorithm (SHA). ASC X9 Secretariat: American Banker's Association; 1993.

Bellovin SM, Merritt M. Encrypted key exchange: password-based protocols secure against dictionary attacks. In: Proceedings of IEEE computer society symposium on research in security and privacy; May 1992. p. 72—84.

<http://www.cs.berkeley.edu/~mori>; 2004 [accessed on October 08, 2004].

Goyal V. How to re-initialize a hash chain, Cryptology ePrint Archive, Report 2004/097. Available from: eprint.iacr.org; 2004a.

Goyal V. Password based authentication without public key cryptography, manuscript; April 2004b.

Goyal V, Kumar V, Singh M, Abraham A, Sanyal S. CompChall: addressing password guessing attacks. In: IEEE international conference on information technology: coding and computing (ITCC'05). USA: IEEE Computer Society; 2005. p. 739—44.

Hackers find new way to bilk eBay users, CNET%20news.com; March 25, 2002.

Halevi Shai, Krawczyk Hugo. Public key cryptography and password protocols. ACM Transaction on Information and System Security August 1999;2(3):230—68.

Klein DV. Foiling the cracker: a survey of, and improvements to password security. In: Proceedings of the second USENIX UNIX security workshop; 1990. p. 5—14.

Lamport L. Password authentication with insecure communication. Communications of the ACM 1981;24:770—2.

Morris R, Thompson K. Password security: a case history. Communications of the ACM November, 1979;22(11):594—7.

Benny Pinkas, Tomas Sander. Securing passwords against dictionary attacks. In: Proceedings of the ninth ACM conference on computer and communications security; 2002.

Ritter Terry. Salts. Online article at, http://www.ciphersbyritter.com/NEWS6/SALT.HTM; November, 1979.

Rivest R. Can we eliminate certificate revocations lists? In: Financial cryptography. LNCS. Springer; 1998. p. 178—83.

Salt (cryptography), wikipedia, the free encyclopedia. Available from: http://en.wikipedia.org/wiki/Salt_(cryptography); 1998.

Vipul Goyal is currently a Cryptography Consultant at OSP Global, LLC. His primary areas of interest are cryptography system design with a current focus on authentication protocols, certificate revocation and structures generated from hash functions like hash chains, hash trees and one time digital signatures. He has several international publications in these areas.

Virendra Kumar is a student at Institute of Technology, Banaras Hindu University, India. His research interests include cryptography and cryptanalysis, design of secure identification and authentication protocols, fair-exchange and secure multi-party computation.

Mayank Singh is a student at Institute of Technology, Banaras Hindu University, India. His research interests include cryptography and cryptanalysis, design of secure identification and authentication protocols, fair-exchange and secure multi-party computation.

Ajith Abraham currently works as a Distinguished Visiting Professor under the South Korean Government's Institute of Information Technology Assessment (IITA) Professorship programme at Chung-Ang University, Korea. His primary research interests are in computational intelligence with a focus on using evolutionary computation techniques for designing intelligent paradigms. Application areas include several real world knowledge-mining applications like information security, bioinformatics, Web intelligence, energy management, financial modelling, weather analysis, fault monitoring, multi criteria decision-making, etc. He has authored/co-authored over 200 research publications in peer reviewed reputed journals, book chapters and conference proceedings of which three have won 'best paper' awards.

He is the Editor of The International Journal of Hybrid Intelligent Systems (IJHIS), IOS Press, Netherlands; Journal of Information Assurance and Security (JIAS), USA; International Journal of Computational Intelligence Research (IJCIR), Neurocomputing Journal, Elsevier Science, The Netherlands; International Journal of Systems Science (IJSS), Taylor & Francis, UK; Journal of Universal Computer Science (J.UCS), Austria; Journal of Information and Knowledge Management, World Scientific, Singapore; Journal of Digital and Information Management (JDIM), Digital Information Research Foundation, India and International Journal of Neural Parallel and Scientific Computations (NPSC), USA. Since 2001, he is actively involved in the Hybrid Intelligent Systems (HIS) and the Intelligent Systems Design and Applications (ISDA) series of annual International conferences. He was also the General Co-Chair of The Fourth IEEE International Workshop on Soft Computing as Transdisciplinary Science and Technology (WSTST05), Japan and the Program Co-Chair of the Inaugural IEEE Conference on Next Generation Web Services Practices, Seoul, Korea. He received PhD degree from Monash University, Australia. More information at http://ajith.softcomputing.net

Sugata Sanyal is in the Faculty of the Tata Institute of Fundamental Research, India. He received his Ph.D. degree from Mumbai University, India, M.Tech from IIT, Kharagpur, India and B.E. from Jadavpur University, India. His current research interests include security in wireless and mobile ad hoc networks, distributed processing, and scheduling techniques. He has published numerous papers in national and international journals and attended many conferences. He is in the editorial board of three International Journals. He is co-recipient of ''Vividhlaxi Audyogik Samsodhan Vikas Kendra Award (VASVIK)'' for Electrical and Electronics Science and Technologies (combined) for the year 1985. He was a Visiting Professor in the Department of Electrical and Computer Engineering and Computer Science in the University of Cincinnati, Ohio, USA in 2003. He delivered a series of lectures and also interacted with the Research Scholars in the area of Network Security in USA, in University of Cincinnati, University of Iowa, Iowa State University and Oklahoma State University. He has been an Honorary Member of Technical Board in UTI (Unit Trust of India) and SIDBI (Small Industries Development Bank of India). He has also acted as a consultant to a number of leading industrial houses in India. More information about his activities is available at http://www.tifr.res.in/~sanyal.