Differentiated Virtual Passwords, Secret Little Functions, and Codebooks for Protecting Users From Password Theft

Yang Xiao, *Senior Member, IEEE,* Chung-Chih Li, Ming Lei, and Susan V. Vrbsky

***Abstract*—In this paper, we discuss how to prevent users' passwords from being stolen by adversaries in online environments and automated teller machines. We propose differentiated virtual password mechanisms in which a user has the freedom to choose a virtual password scheme ranging from weak security to strong security, where a virtual password requires a small amount of human computing to secure users' passwords. The tradeoff is that the stronger the scheme, the more complex the scheme may be. Among the schemes, we have a default method (i.e., traditional password scheme), system recommended functions, user-specified functions, user-specified programs, and so on. A function/program is used to implement the virtual password concept with a tradeoff of security for complexity requiring a small amount of human computing. We further propose several functions to serve as system recommended functions and provide a security analysis. For user-specified functions, we adopt secret little functions in which security is enhanced by hiding secret functions/algorithms.**

***Index Terms*—Codebooks, differentiated virtual passwords, key logger, phishing, secret little functions, shoulder-surfing.**

I. Introduction

TODAY, THE Internet has entered into our daily lives as more and more services have been moved online. Besides reading the news, searching for information, and other risk-free activities online, we have also become accustomed to other risk-related work, such as paying using credit cards, checking/composing emails, online banking, and so on. While we enjoy its convenience, we are putting ourselves at risk. Most current commercial websites will ask their users to input their user identifications (IDs) and corresponding passwords for authentication. Once a user's ID and the corresponding password are stolen by an adversary, the adversary can do anything with the victim's account, which can lead to a disaster for the victim. As a consequence of increasing concerns over such risks, protecting users' passwords on the web has become increasingly critical.

The secure protocol SSL/TLS [1] for transmitting private data over the web is well-known in academic research, but most current commercial websites still rely on the relatively weak protection mechanism of user authentications via a plaintext password and user ID. Meanwhile, even though a password can be transferred via a secure channel, this authentication approach is still vulnerable to the following attacks: 1) in *phishing attacks*, phishers attempt to fraudulently acquire sensitive information, such as passwords and credit card details, by masquerading as a trustworthy person or business in an electronic communication [2]; 2) *Password Stealing Trojan* programs contain or install malicious codes. Examples include: a) key loggers capturing keystrokes in the machine; and b) Trojan Redirectors redirecting end-users network traffic to a desired location [5]; and 3) *Shoulder Surfing* steals others' sensitive personal information by looking over victims' shoulders [12], [16], [20] or capturing users' inputs and screens by taking pictures and videos using cameras and video recorders, respectively. Many schemes, protocols, and software have been designed to prevent users from some specified attacks. However, to the best of our knowledge, there is not a scheme which can defend against all the attacks listed above at the same time.

In this paper, we present a password protection scheme that involves a small amount of human computing in an Internet-based environment or a ATM machine, which will be resistant to phishing scams, Trojan horses, and shoulder-surfing attacks. We propose a virtual password concept involving a small amount of human computing to secure users' passwords in online environments. We propose differentiated security mechanisms in which a user has the freedom to choose a virtual password scheme ranging from weak security to strong security. The tradeoff is that stronger schemes are more complex. Among the schemes, we have a default method (i.e., traditional password scheme), a system recommended function, a user-specified function, a user-specified program, and so on. A function/program is used to implement the virtual password concept by trading security for complexity by requiring a small amount of human computing. We further propose several functions to serve as system recommended functions and provide a security analysis. We analyze how the proposed schemes defend against phishing, key logger, shoulder-surfing, and multiple attacks. In user-specified functions, we adopt secret little functions in which security is enhanced by hiding

Manuscript received June 21, 2011; revised October 24, 2011 and December 8, 2011; accepted December 31, 2011. Date of publication February 15, 2012; date of current version May 22, 2014. This work was supported in part by the National Science Foundation, under Grants CCF-0829827, CNS-0716211, CNS-0737325, and CNS-1059265.

Y. Xiao and S. V. Vrbsky are with the Department of Computer Science, University of Alabama, Tuscaloosa, AL 35487 USA (e-mail: yangxiao@ieee.org; vrbsky@cs.ua.edu).

C.-C. Li is with the School of Information Technology, Illinois State University, Normal, IL 61790 USA (e-mail: cli2@ilstu.edu).

M. Lei is with Oracle Corporation, Redwood City, CA 94065 USA.

Digital Object Identifier 10.1109/JSYST.2012.2183755

secret functions/algorithms. To the best of our knowledge, our virtual password mechanism is the first one which is able to defend against all three attacks. We further propose a scheme to adopt μTESLA to be used for re-keying and to defend against phishing.

The proposed functions include secret little functions and two other schemes called codebook and reference switching functions. Our objective is to produce a function achieving both: 1) ease of computation; and 2) security. However, since simplicity and security conflict, it is difficult to achieve both.

The idea of this paper is to add some complexity, through user computations performed by heart/hand or computation devices, to prevent the three kinds of attacks. There is a tradeoff of how complex the computation by the users can be. One goal is to find an easy to compute but secure scheme for computing.

We believe that, for some sensitive accounts such as online bank accounts and online credit card accounts, users are likely to choose additional complexity which requires some degree of human computing in order to make the account more secure.

The rest of this paper is organized as follows. We describe related work about password protection in Section II. In Section III, we propose the idea of the virtual password, differentiated security mechanisms, and user-specified functions or programs, in which we propose the concept of secret little functions. Two functions (the codebook approach and reference switching approach) are proposed in Section IV. We provide some quantitative analysis in Section V. In Section VI, we describe implementation issues of our scheme. Finally, we conclude our paper and describe our future work in Section VII.

II. Related Work

How to shield users' passwords from being stolen by adversaries is not a new topic, but it is always important because adversaries keep inventing more and more advanced attacks to break the current defense schemes. This results in more research on protecting users from such attacks. In this section, we briefly introduce the previous work on defending against user password-stealing attacks for the three major categories.

Phishing attacks are relatively new but very effective. There are two typical types of phishing. First, to prevent phishing emails [27], [29], [30], a statistical machine learning technology is used to filter the likely phishing emails; however, such a content filter does not always work correctly. Blacklists of spamming/phishing mail servers are built in [31] and [32]; however, these servers are not useful when an attacker hijacks a virus-infected PC. In [11], [24], and [25], a path-based verification was introduced. In [14], a key distribution architecture and a particular identity-based digital signature scheme were proposed to make email trustworthy. Second, to defend against phishing websites, the authors in [21] and [33] developed some web browser toolbars to inform a user of the reputation and origin of the websites which they are currently visiting. In [6]–[10], the authors implemented password hashing with a salt as an extension of the web browser [6], [9], [10], a web proxy [13], or a stand-alone Java Applet [15]. Regardless of the potential challenges considered in an implementation, such password hashing technology has a roaming problem because not every web browser installs such an extension or sets the web proxy. Another more important challenge is that more web browsers need to be designed in which designers are not reluctant to include specified extensions for each other.

Unlike phishing, malicious Trojan horses, such as a key logger, are not attacks, and sophisticated users can avoid them. Such programs are also easy to develop [17], and there is a great deal of freeware that can be downloaded from the Internet to prevent them. You may be advised to install an anti-spyware or anti-virus software package on your machine or to set up a firewall to block suspicious packages from the outside. However, in the event you are traveling to some place without carrying your own computer and you have to seek help from an Internet cafe to access the Internet, do you want to trust the computers in the Internet cafe? In [17], the author presented a tricky method which can confuse a keylogger, which works as follows. Instead of typing your whole password into the login field, the user changes focus outside the login form and types some random characters between any two successive password characters. However, this trick does not shield the user from keylogger attacks. It only makes it slightly more difficult because it is very easy to record all the keys, mouse events, and applications of the focus. The authors in [18] and [19] used a virtual pad for the login system, which allows a user to click the virtual keyboard on the screen instead of typing on the physical keyboard, but such a virtual keyboard faces some of the same problems as above (i.e., an adversary can record all the mouse events with a combination of screen snapshots to figure out what the user clicks on the screen).

Alphanumeric password systems are easily attacked by shoulder-surfing, in which an adversary can record the user motions by a hidden camera when the user types in the password. In [22], the authors adopted a game-like graphical method of authentication to combat shoulder-surfing; it requires the user to pick out the passwords from hundreds of pictures, and then complete rounds of mouse-clicking in the Convex Hull. However, the whole process needs the help of a mouse and it takes a long time. In [23], the authors proposed a scheme to ask a user to answer multiple questions for each digit. In this way, it is only somewhat resistant to shoulder-surfing because, if an adversary catches all the questions, then they will know what the password is. In [23], a game-based method was designed to use cognitive trapdoor games to achieve a shield for shoulder-surfing. The author in [26] filed a patent to allow a user to make some calculations based on a system generated function and random number for the user to prevent password leaking. However, the scheme in [26] is not anti-phishing and the password can be stolen if an adversary uses a camera to record all the screens of the system and motions of the victim.

Note that using SSL/TLS in websites could not prevent key logger attacks.

In the traditional challenge-response protocol: 1) a user sends his/her identity to the server, who generates and returns a random number r, as the challenge; and 2) the user's response is $f(r, h(P))$, where $f()$ is a function and $h()$ is the hash function, and P is the password. However, the above challenge-response protocol suffers phishing attacks, Password Stealing Trojan programs (such as key loggers and Trojan Redirectors), and Shoulder Surfing. For example, both key logger and Shoulder Surfing can steal user's password. A zero-knowledge password is a way to verify that a user has a certain secret information. However, such secret information once typed in the system suffers key logger and Shoulder Surfing attacks.

In the previous sections, we have briefly introduced some schemes without including methods which need hardware support. None of the schemes above can prevent phishing, Trojan horse, and shoulder-surfing at the same time.

A one-time password (OTP) does not use a static password, and therefore can prevent replay attacks. There are several approaches to generate and distribute OPTs: 1) a time-synchronization method; 2) a mathematical algorithm to generate a password based on old ones; and 3) a mathematical algorithm to generate a challenge [36], [37]. In the time-synchronization method, a password is generated based on the current time, while the server and the client programs need time-synchronization via a physical electronic token. In the second method, OPTs must be used by a predefined order such as using a one-way hush function [36], [37]. In the third method, challenge-response can be done by a physical token. Therefore, an OPT method needs a electronic token, mobile phone, or out-of-band channel such as SMS messaging [36]. The idea of our proposed virtual password is similar to OPT. However, the approaches are different. For example, in the proposed secret little function approach, the secret little function is built between the server and the user (a human being), whereas in a OPT method, communications are done between the server and a physical device (either a token, a computer, or a cellular phone). Also, a OPT assumes that the physical device is secure and not compromised; whereas our secret little function method does not need such an assumption. For example, when a user on a business trip uses a un-trusted machine at an Internet Cafe, or a ATM machine, a OPT method becomes difficult to used since the physical device (token or cellular phone) is difficult to connect via the un-trusted machine or the ATM machine. But our method can be easily and safely used.

Note that an early short version of this paper was presented in the conference [35]. There is also some related work for password and authentication methods [38]–[68].

III. Differentiated Virtual Passwords and Secret Little Functions

In Section III-A, we propose the idea of the virtual password. In Section III-B, we propose differentiated security mechanisms in which a user has the freedom to choose a virtual password scheme ranging from weak security to strong security. In Section III-C, we propose user-specified functions or programs, in which we propose the concept of secret little functions. We discuss virtual password function with a helper-application in Section III-D. We discuss virtual password functions without a helper-application in general in Section III-E. We present μTESLA authentication in Section III-F

A. Virtual Password

To authenticate a user, a system (S) needs to verify a user (U) using the user's password (X) and ID (also denoted as U) which the user provides. In this procedure, S authenticates U by using U and X, which is denoted as: $S \rightarrow U$: U, X. Both U and X are fixed. It is reasonable that a password should be constant so that it can be easily remembered. However, the price of being easily remembered is that the password can be stolen by others and then used to access the victim's account. At the same time, we can not put X in a randomly variant form because it would be impossible for a user to remember the password. To confront such a challenge, we propose a scheme using the new concept of virtual password.

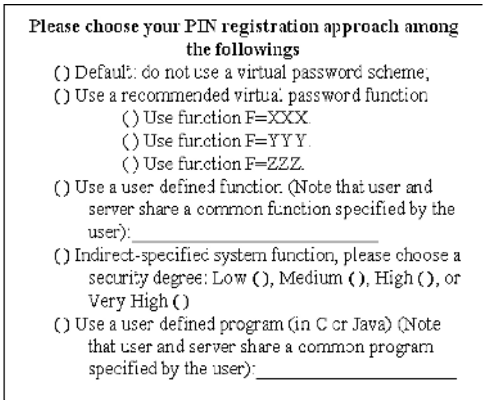
A *virtual password* is a dynamic password that is generated differently each time from a *virtual password scheme* and then submitted to the server for authentication. A virtual password scheme P is composed of two parts, a fixed alphanumeric X (i.e., the real password, also called the hidden password) and a function F from the domain ψ to ψ, where the ψ is the letter space which can be used for passwords. Since we call $P = (X, F)$ a virtual password scheme, we call F a virtual password function (VPF). The result (denoted as V) of the VPF is called a virtual password, and F may have some hidden parameters, H, which are the secrets between the server and the user. If this is the case, we denote F with $F_H(...)$. Note that the VPF can be a secret between the server and the user. Let $X = x_1x_2...x_n$ denote a vector standing for the hidden password, where x_i $(i = 1...n)$ is a digit, and n is the length of the vector. Let $R = r_1r_2...r_n$ denote the random number provided by the server, also called the random salt, and prompted in the login screen by the server. $V = v_1...v_n$ is the virtual password used for authentication. The user input includes (U, V), where U is the user ID. On the server side, the server can also calculate V in the same way to compare it with the submitted password. We use $V = F_H(X, R)$ or $F_H(x_i, r_i) = v_i$ interchangeably in the rest of this paper.

It is easy for the server to verify the user if F is a bijective function. If F is not a bijective function, it is also possible to allow the server to verify the user as follows. The server first finds the user's record from the database based on the user's ID (i.e., U), then it computes V and compares it with the one provided by the user. A bijective function makes it easier for the system to use the reverse function to deduce F's virtual password. Therefore, we do not assume that F is a bijective function.

The user should be free to pick the hidden password. We propose a differentiated security mechanism in the next subsection to allow the user to choose a VPF.

B. Differentiated Security via a VPF

We have introduced the concept of the virtual password; next, we detail how to apply it in an Internet-based environment. We propose a differentiated security mechanism for



Please choose your PIN registration approach among the followings

() Default: do not use a virtual password scheme;

() Use a recommended virtual password function

() Use function F=XXX.

() Use function F=YYY.

() Use function F=ZZZ.

() Use a user defined function (Note that user and server share a common function specified by the user):____________________

() Indirect-specified system function, please choose a security degree: Low (), Medium (), High (), or Very High ()

() Use a user defined program (in C or Java) (Note that user and server share a common program specified by the user):____________________

Fig. 1. Screenshot of user registration: Step 1.

system registration in which the system allows users to choose a registration scheme ranging from the simplest one (default) to a relatively complex one, where a registration scheme includes a way to choose a virtual password function. The more complex the registration, the more secure the system is, and the more user involvement is required. A screenshot of the first step of the proposed registration is shown in Fig. 1. Whether a virtual password scheme is used or not, the user is required to input the read password and ID in Step 2 of the registration.

In Fig. 1, a user has the freedom to choose a default approach in the traditional way or a more complex scheme as proposed in this paper. A user can choose a recommended virtual password function, define his/her own virtual password function, or even define a common program to share between the user and the server to calculate the password.

1) The system recommended approach is that, after the system receives a registration request, it automatically generates a function. The users do not have to provide extra information about the function to the server except for some necessary parameters, called hidden parameters (H).
2) The user specified function approach is the one in which users themselves can choose any function they like. However, such freedom is based on the assumption that the user has some basic knowledge about VPFs, which can be introduced by an online introduction.
3) The indirectly-specified approach, instead of letting either the user or the server make the full decision, allows a user to specify the desired security degree. Then the server will assign a function according to that degree.
4) An extreme scheme is that the user can even provide a program in C or Java instead of a function. This requires a very advanced user.

Note that, except for the default approach, either human computing is involved or a handhold device (or a computer) which can be programmed to compute the virtual password is needed. We could develop a smart application to make the complex calculation for the user which can be run on the mobile device, such as a cellular phone, PDA, smart phone, iphone, personal computer, or programmable calculator, to relieve the user from complicated calculations and to overcome any short-term memory problem. If such a helper-application is involved, we should make sure that the helper-application itself is unique to each user account and only works for the corresponding user account.

Regardless of the approach chosen, a user's registration in the system is similar (i.e., the user submits a user ID and a fixed password). The one difference from a traditional approach is that in the virtual password scheme, a VPF must be set during the registration phase.

The server then delivers this function information to the user via some channels, such as displaying it on the screen or in an email. The user needs to either remember this function together with the password they have chosen or to save them in disks or emails. The user-specified password and the system-generated function are combined to form a virtual password scheme.

We also note that a small amount of human-computing is involved in the authentication process. We have to choose a VPF to make the calculation as simple as possible if the helper-application is not used. A user has to remember both the hidden password and the function (i.e., VPF), and as a result more effort is required to remember them. However, the virtual password will be resistant to a dictionary attack, mostly because users like to create a password which is either related to their own name, date of birth, other simple words, and so on.

In a traditional password scheme, users can change their password. This is also true in our virtual password scheme. Unlike the traditional scheme, users can change the hidden password, the VPF, or both.

C. User-Specified Functions/Programs

The strongest security approaches let the user define a user-specified function or program. Since the chosen function is only known by the server and the user and the key space of functions are infinite with high-order, these approaches are very secure for even simple functions.

In many classical ciphers, secret encryption algorithms are common. In modern ciphers, encryption algorithms are open to the public but keys of these algorithms are kept secret. One reason that modern ciphers seldom choose secret encryption algorithms is that secret encryption algorithms prevent communication among parties such as commercial products, networking protocols, and so on. Therefore, the approach in which only keys are kept as secrets (small data) and algorithms (large programs) are open to the public for implementation is very popular in modern ciphers.

The reason for using secret encryption algorithms (i.e., user-specified VPFs) is that secrets are very personal to a particular user and should not be known by others except the server. On the other hand, for example, a wireless local area network (WiFi) needs open encryption algorithms to allow products from different companies to communicate with each other. Otherwise, one company's WiFi card could not communicate with that of another company. However, in our application, communication is only between a user and a server so that it is good to use secret encryption algorithms, since secret encryption algorithms enhance security by hiding the algorithms/functions. Therefore, we claim that, for a very personal communication, such as one between a user and a server, it is acceptable to use secret encryption algorithms. The function space is infinite with high-order.

Some people may be concerned that trained professionals cannot provide an easy and secure function, so most users may not either. This concern is not actually necessary. Since even a very simple function will be secure because the attackers do not know what kind of functions the user chose (i.e., functions are kept as secrets instead of keys and the resulting function space is infinite with high-order). Examples of simple functions can be as follows:

1) flip one bit in the password;
2) flip one digit in the password;
3) add one to each odd digit and minus one in each even digit;
4) the first digit of the password is tripled; 100x+birthdate, where x is the real password in an integer form transferred from ASCII codes;
5) reversing even bits of the real password in a binary form;
6) a secret function x+a, where a is a constant, and x is the fixed password;
7) a secret function x-a, where a is a constant, and x is the fixed password;
8) a secret function a*x, where a is a constant, and x is the fixed password;
9) add an additional constant digit/character at a fixed place;
10) a secret linear function [a(x+y)+c] mod (m), where a and c are constants, y is a random number generated by the server, and x is the fixed password;
11) and so on.

User specified functions can be infinite. Since attackers do not know the function forms (i.e., secret encryption algorithms), these simple functions are very secure. Otherwise, it would be easy to attack these functions. Note that user-specified functions do not need to be bijective.

We call these simple and secure functions *secret little functions*. They are useful in our context. One problem is that extra effort is required in programming the function into the server upon the creation of an account, so human intervention that may be needed.

Another constraint is that secret little functions must use the random number provided by the server; otherwise, it would still be subject to Key-logger attacks since the attackers do not need to know the function but can simply input the same capture inputs again to gain access.

Advanced users can also define a program to be used.

As for the server storage, assuming that each VPF needs 0.2K storage space, the total storage space for 10 000 user accounts registered on the server is 2M, which is reasonable.

Note that the secret little function is that the function F defined in $P = (X, F)$.

D. VPF With a Helper-Application

If a helper-application is available for the user, the user needs to type the random salt into the helper-application; subsequently, the virtual password is generated by the helper-application. The user then types the generated virtual password in the login screen. In this way, the extra time required is very small and the precision will be 100% correct as long as the user types the correct random salt displayed on the login screen.

This works when the user has a mobile device, such as a cellular phone, PDA, smart phone, or iphone. However, such mobile devices are not able themselves to communicate with the server to which the user wants to login. No matter how complex the VPF is, the helper-application can always generate the correct virtual password for the user. This case is the most sophisticated one, and it is also the most convenient approach for the user.

For password changing, the user only needs to get a new helper-application after the password change instead of remembering all the changed parts of the virtual password. Note that the server must make the corresponding changes too.

A one-way hash function and many other functions (such as known encryption algorithms) can serve as VPFs.

If we further assume that the helper-application can communicate with the server, the user only needs to type the random salt in the helper-application, and then the rest of the work is done by the helper-application. The helper-application can generate the virtual password and submit the login request associated with the user account information, which can be built into the helper-application for the corresponding user. For password changing, the helper-application communicating with the server is a better way to change passwords and make them more secure (i.e., the helper-application can periodically make the password change request to server and update the corresponding virtual password built into the helper-application). The whole process can be completely transparent to the user.

E. VPF Without a Helper-Application

If there is no helper-application for a user, the user needs to calculate the virtual password from the VPF with the inputs, the random salt, and the hidden password. The whole login process may take a little bit longer because it requires the user to perform some calculations. This must work for the user who has no mobile device, so the VPF should not be too complicated for human computing.

Password changing in VPF is similar to traditional password changing. The user can choose a new password, which is the hidden password, a new VPF, or both. After such changes, the user needs to remember the new virtual password.

The VPF plays a critical role in the virtual password, especially when the user chooses the option of "Use a recommended virtual password function" in Fig. 1. There are an infinite number of VPFs, so designing an appropriate function is very critical to the success of our scheme.

In order to defend against phishing, key-loggers, and shoulder-surfing while the system is authenticating the user, this function should meet the following criteria.

1) The function should have some random input provided by the server, which then allows the users to type in different inputs each time they log in the system. This ensures that the key logger can not steal the password

because the real password is not typed and the typed inputs change each time.

2) The function should be easy for the users. To make the system more secure, we could increase the complexity of the VPF. However, the resulting function may be very difficult to remember or utilize. The objective is to design less complex but secure VPFs.
3) The function should be *unobservable* (i.e., the observed password the user types in for the login session does not disclose hidden secrets), so that adversaries cannot use the stolen information to log into the system.
4) The function should be *unsolvable* (i.e., the adversaries should not be able to solve the function with the potential information they are able to obtain).

These four requirements are used to guide us to design the appropriate VPFs. There are many functions which meet all the requirements listed above.

F. μTESLA Authentication

In this subsection, we propose a scheme to adopt μTESLA to be used for re-keying and defending against phishing.

In the previous sections, we discussed how to use the virtual password to defend against phishing, key loggers, and shoulder-surfing. In this section, we propose another scheme to guard against phishing attacks by allowing the user to authenticate the server and adopting μTESLA to provide freshness of the server key. The purpose of this scheme is that it can be used for authentication of the server before re-keying of the previous scheme. This μTESLA scheme can be very useful when the web browser or other client side applications, such as the helper-application in our virtual password, can have an authentication function implemented. This scheme can also be used to protect from phishing via emails.

μTESLA [34] is an authentication scheme which was originally designed for sensors to authenticate a broadcast message sender in a sensor network based on a public one-way hash function F.

We could use the methodology to defend against phishing attacks or to authenticate the server before re-keying. We adjust it with the assumption that the server side and client side will choose the same public hash function; we discuss how it works in the registration phase, sign-on phase, and password change phase.

In the registration phase, upon a registration request, in addition to preparing the general password, user id, and other information, the server needs to generate a chain key $K_m, ..., K_0$ by randomly choosing the K_m and then producing the $K_{n-1} = F(K_n)$ where $n = 0, 1, ..., m$. The server will then pass the K_0 to the client.

In the sign-on phase, once the sign-on request arrives at the server side, the server presents the sign on screen to the clients who need to provide the authentication code, which will be encrypted by the latest key. For example, the nth time the user signs in to the system, the server produces the authentication code as $E_{K_{n-1}}(K_n)$ and passes this to the client. When the client receives the package, they first need to decrypt the authentication code with the current key (i.e., K_{n-1}) then to get the K_n and then use this K_n to verify this sign-on screen is from the right server in the following way: if $F(K_n) = Key$, it is verified and the currently held key is updated to be K_n; otherwise, it is denied.

In this way, the client can verify the server and be protected from phishing attacks because the phisher has no knowledge of the $K_0, ..., K_m$ and therefore is not able to fake the authentication code. Furthermore, the server should use the latest used key to encrypt the current key, since, if the authentication code is not encrypted, the phisher could pretend that they are clients and try to log in to the system. The server then presents the login screen along with the new authentication code K_n, which makes if difficult for the phisher to fake a login screen with the correct authentication code to lure the client.

Because the number of keys from K_0 to K_m is finite, the server will eventually use up all the authentication codes, even if we choose a very large value of m. The server and client should build a scheme to regenerate their authentication code, which we refer to here as re-keying of the authentication key refreshes. This is an easy job and can be conducted once both the client and the server verify each other. The server needs to generate a new chain of keys, as it did in the user registration phase, and to deliver the first of the keys to the client.

This μTESLA scheme will work effectively to shield the clients from phishing attacks, and it could be used together with our virtual password scheme to protect the user's password.

IV. Codebook and Reference Switching

Here, we propose two approaches to the virtual password function. They are not perfect, but they are acceptable in the hostile password phishing environment. For the first approach, some small codebooks will be needed. A codebook should be small enough to be printed on a pocket-sized card, stored in a saved e-mail, or stored in a PDA or cell phone for the user to carry. It is not impossible but would be unrealistic to ask the user to remember the entire codebook. For the second approach, we design a function that is easy to compute with paper-and-pencil or a nonscientific calculator. However, its inverse function should be difficult to compute without knowing some hidden parameters. Our ultimate goal is to design a zero-knowledge interactive proving protocol, but this is impossible given the constraints mentioned earlier. Thus, our next ideal functions would be those that do not give away enough information to significantly compromise the user's account.

A. Codebook

We first assume that our server has sufficient computing power to run a cryptographically secure random number generator (RNG). This requirement is necessary to protect the whole system; in case a user loses their codebook, the system will not be compromised and the user can easily ask for a new codebook without changing the parameters of the RNG. Note that linear congruential generators is not as cryptographically secure RNG.

Our first codebook is rather straightforward. In the setup session, the user decides the length of the password, n. The server then gives n 10-digit random numbers. Suppose that we are doing this to protect a 4-digit PIN (i.e., $n = 4$). The sever outputs four random numbers, R_0, R_1, R_2, and R_3, with each having ten digits. Let $r_{(i,0)}, r_{(i,1)}, r_{(i,2)}, ..., r_{(i,9)}$ denote the ten digits of R_i. The user's codebook is given as follows:

$r_{(0,0)}$	$r_{(0,1)}$	$r_{(0,2)}$	...	$x_{(0,9)}$
$r_{(1,0)}$	$r_{(1,1)}$	$r_{(1,2)}$	...	$x_{(1,9)}$
$r_{(2,0)}$	$r_{(2,1)}$	$r_{(2,2)}$	...	$x_{(2,9)}$
$r_{(3,0)}$	$r_{(3,1)}$	$r_{(4,2)}$	...	$x_{(3,9)}$

It is up to the user to decide how to store or memorize the codebook. To login to the system, the system will present a 4-digit random number $R = abcd$, where each letter represents a digit. The virtual password the user must key in would be $v_{(i,a)}v_{(i,b)}v_{(i,c)}v_{(i,d)}$.

For security analysis, we consider the phishing attack because it is the most aggressive attack in which the adversary can control the random number R. For each attack, the phisher will provide a fake random number R to the victim. If successful, the adversary will get four corresponding digits in the codebook. As a result, the chance that the adversary can correctly guess one digit of the password is the chance that the system asks for the same position plus the chance that the system asks for the other nine positions and the adversary guesses them correctly (i.e., $\frac{1}{10} + \frac{9}{10} \times \frac{1}{10} \approx \frac{1}{5}$).

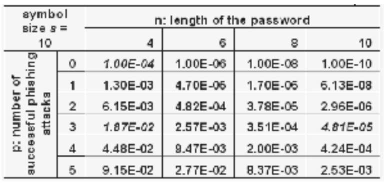
In other words, the chance of the adversary breaking into the victim's account after one successful phishing is $(0.2)^4 = 1/625$. It is likely that the adversary will conduct more than one attack and that the victim will not be aware of the situation in the first few rounds of phishing. To maximize the information gained, the adversary will ask different positions in each phishing. Let p be the number of successful phishing attacks on the same user. Also, let n be the length of the password and s the number of different symbols for a digit (in the present example, $s = 10$). We have the following formula for the chance the adversary may get into the victim's account:

$$\left(\frac{p}{s} + \frac{s-p}{s} \times \frac{1}{s}\right)^p = \left(\frac{1+p}{s} - \frac{p}{s^2}\right)^p. \qquad (1)$$

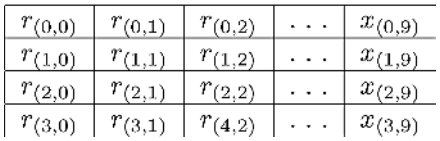
Table I contains the results from using this codebook. Conventionally, we use four digits of Arabic numbers (symbol size $s = 10$) for a PIN code. Under a phishing free environment, the code is protected by its key space of size 104. Without this virtual password protection, one successful phishing attack will completely invalidate the PIN code. It is clear that our codebook approach can significantly decrease the chance of breaking the protection after a few successful phishings. It is safe to assume that, after a few phishing attacks, the victim will become suspicious and stop responding to the phisher. According to the table above, after three successful attacks, the adversary's odds of getting into the system increases to 1.87×10^{-2}. In this case, the victim's account is still relatively safe if we require the server to lock the account after multiple attempts to login to the account with an invalid password. We may also increase the length of the password to resist more attacks. For example, if we use ten digits, the chances of compromising the account after three successful phishing attacks are about the same as a 4-digit PIN code under a phishing-free environment. However, we should take every precaution and assume that five or more successful phishing attacks can be made on a careless user. To have a password with 20 digits is unrealistic. Instead, we can increase the symbol size by allowing letters and some special symbols to be used in the passwords. In practice, 64 is a reasonable symbol size. We have the results in Table II.

TABLE I

CHANCE OF BREAKING THE PASSWORDS UNDER PHISHING ATTACKS (SYMBOL SIZE = 10)



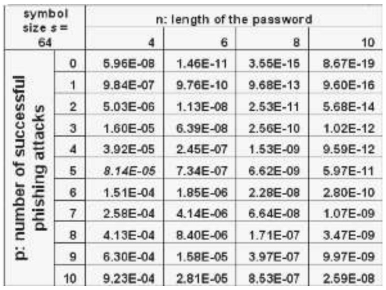
symbol size s = 10		n: length of the password			
		4	6	8	10
p: number of successful phishing attacks	0	1.00E-04	1.00E-06	1.00E-08	1.00E-10
	1	1.30E-03	4.70E-05	1.70E-06	6.13E-08
	2	6.15E-03	4.82E-04	3.78E-05	2.96E-06
	3	1.87E-02	2.57E-03	3.51E-04	4.81E-05
	4	4.48E-02	9.47E-03	2.00E-03	4.24E-04
	5	9.15E-02	2.77E-02	8.37E-03	2.53E-03



$r_{(0,0)}$	$r_{(0,1)}$	$r_{(0,2)}$	...	$x_{(0,9)}$
$r_{(1,0)}$	$r_{(1,1)}$	$r_{(1,2)}$	...	$x_{(1,9)}$
$r_{(2,0)}$	$r_{(2,1)}$	$r_{(2,2)}$	...	$x_{(2,9)}$
$r_{(3,0)}$	$r_{(3,1)}$	$r_{(4,2)}$	...	$x_{(3,9)}$

TABLE II

CHANCE OF BREAKING THE PASSWORDS UNDER PHISHING ATTACKS (SYMBOL SIZE = 64)



symbol size s = 64		n: length of the password			
		4	6	8	10
p: number of successful phishing attacks	0	5.96E-08	1.46E-11	3.55E-15	8.67E-19
	1	9.84E-07	9.76E-10	9.68E-13	9.60E-16
	2	5.03E-06	1.13E-08	2.53E-11	5.68E-14
	3	1.60E-05	6.39E-08	2.56E-10	1.02E-12
	4	3.92E-05	2.45E-07	1.53E-09	9.59E-12
	5	8.14E-05	7.34E-07	6.62E-09	5.97E-11
	6	1.51E-04	1.85E-06	2.28E-08	2.80E-10
	7	2.58E-04	4.14E-06	6.64E-08	1.07E-09
	8	4.13E-04	8.40E-06	1.71E-07	3.47E-09
	9	6.30E-04	1.58E-05	3.97E-07	9.97E-09
	10	9.23E-04	2.81E-05	8.53E-07	2.59E-08

According to the above table, if the symbol size is increased to 64, the security level of four digit passwords after five successful phishing attacks is still at the level of conventional 4-digit PIN codes under a phishing-free environment. In practice, it is not likely that a user will respond to the phisher more than five times without getting suspicious. Note that our concern is very different from the chosen (or known)-plain text attack in the context of cryptography because a large amount of plain-cipher text is not available to the phisher.

B. Reference Switching

Let Σ denote the alphanumeric set, where alphabets are case insensitive. Each element in Σ is coded by $0 = 0, 1 = 1, ..., 9 = 9, A = 10, B = 11, ...,$ and $Z = 35$ for arithmetic. Here, we present a reference switching (RS) function, $\Sigma* \rightarrow \Sigma*$, using the hidden password as a reference. RS does not have to be bijective, as we believe that this restriction will help the adversary narrow down the possibilities. Since the server and the user share the same hidden function parameters, the two parties can obtain the same result while the adversary has little chance of his/her result agreeing with the server's. Let $X = x_1 x_2 ... x_n$ be the hidden password, provided that we require n to be a prime number not less than 7. This length restriction is required for security reasons that we will explain later. When the user tries to login to the system, the system presents n random alphanumeric digits $R = r_1 r_2 ... r_n$. Let $V = v_1 v_2 ... v_n$ be the virtual password that the user needs to input. $RS(X, R) = V$ is computed as follows. For each $i = 1...n$

$$v_i = r_i x_{(x_i r_i \bmod n)+1} \bmod 36. \qquad (2)$$

Consider the following example. Let $n = 7$, $X = ABCD123$, and S denote the alphanumeric set of x (i.e., $S = A, B, C, D, 1, 2, 3$). Suppose that the sever presents $r = 1\,234\,567$ for the user to compute the virtual password. We have the following calculation:

	x_i	*code*	r_i	$x_i r_i$	x_{a_i}	v_i
x_1	A	10	1	3	D	D
x_2	B	11	2	1	B	M
x_3	C	12	3	1	B	X
x_4	D	13	4	3	D	G
x_5	1	1	5	5	2	A
x_6	2	2	6	5	2	C
x_7	3	3	7	0	A	Y

The fifth column of the table $a_i = (x_i r_i \ mod \ 7) + 1$ is computed for the reference of X that will actually be used in computing v_i. The user should input DMXGACY as the password and this is what the server requires for authentication.

What happens if the user is linked to a phisher's page? Successful phishing, as well as shoulder-surfing and key-logger, will make R and V available to the adversary. Thus, the adversary can recover the set of elements in the sixth column of the table above by $x_{a_i} = r_i^{-1} k_i \bmod 36$. Let such a set be S'; thus $S' = A, B, D, 2$. Clearly, $S' \subseteq S$. However, the positions of these digits will not be revealed no matter how many rounds of successful phishing have been conducted by the adversary because all he/she can get is a subset of S. For example, with $R = 1111111$, the adversary will have $S' = B, C, D, 1, 2, 3$. In the worst case, the adversary may get $S' = S$, but the actual X is still protected to a certain extent by the number of permutations of S. To make this amount more significant, we suggest letting X have at least 11 digits. In this case, even when S is recovered, we still have more than 35 million permutations for the phisher to guess. We suggest letting the length of X be a prime number so that every digit in X has a chance to be used. For example, if the length of X is even and R contains only even digits, then only the digits at even positions in X will be used for V.

V. Quantitative Security Analysis

However, to the best of our knowledge, there is not a scheme which can defend against all the attacks listed above at the same time.

Note that using SSL/TLS in websites could not prevent key logger attacks.

Quantitative analysis is always necessary in security analysis. We differentiate the three common methods of stealing passwords according to the degree of damage they are able to cause. It may be difficult to clearly say that one attack is more powerful than the other, but they do have different properties that may affect design principles for password protection against them.

1) *Phishing.* This is the most aggressive method, since the adversary can choose fake random numbers to help him/her figure out the hidden part of the password. The possible key-space dramatically drops after a few successful phishing attacks. But the adversary cannot conduct successful attacks on the same victim too many times because the victim will eventually become suspicious.
2) *Shoulder-surfing.* This attack is less aggressive due to its physical constraints. The adversary may just randomly pick up a victim and obtain some limited data over the victim's shoulder. Even if a camera is well installed at a certain spot, the victims still arrive on a random basis. The adversary has no control oven them. On the other hand, we may assume that the adversary can also observe the random numbers provided by the system. In other words, if the virtual password function is not complicated enough, the key-space will significantly drop after a few shoulder-surfing attacks.
3) *Key-logger.* This is the least aggressive attack among the three because the adversary cannot control and observe the random number. It may be safe to assume that the adversary cannot observe the random number provided by the system because the random numbers are shown on the screen and user key/mouse actions usually do not react directly to the numbers. However, the victim may not be able to know that he/she is under attack (e.g., a well-hidden Trojan program). Thus, given a sufficiently long period of time, the adversary may collect a certain amount of data for analysis.

Thus, with the strength and weakness of the three possible real-world attacks in mind, we do not have to examine our schemes with theoretical cryptographic standards that may lead us to an impossible task under the computational constraints of designing a virtual password scheme; instead, we look for an economic method that can protect passwords against the three methods under some reasonable assumptions. We propose some criteria as follows.

1) How many successful phishing attacks can be tolerated? We think 10–20 are enough. Tolerating more than 50 or 100 successful phishing attacks may not be necessary because it is not likely that a user will keep giving away information more than that many times without getting any positive feedback. In other words, our schemes should remain secure after 10 to 20 successful phishing attacks. This is already very conservative.
2) For key-logger/shoulder-suffer, our scheme should remain secure after the adversary obtains 500–1000 pairs

of random numbers and virtual passwords. Making it a few thousand attacks may also be possible. A user may login his/her account with the same password 500–1000 times before the system forces the user to change passwords.

Also, the following questions should be answered in order to give a precise analysis.

1) How many tries can an adversary make to login to an account without a correct password? ATM machines usually allow three tries. Some secure band accounts also only allow three tries.
2) What is the size of the key-space we need to maintain after some attacks? So far we have found that it is impossible to maintain sufficiently large key-space after a few attacks (phishing/shoulder-surfing). An alternative is to make it computationally secure; that is, even if there is only one key left consistent with the observed information, the key is still difficult to find. Our RS function may have this property.
3) How many victims can an adversary have?

The three questions will determine the chance of an adversary breaking into an account. The user and the server (for example, the banker) may have different prospects. A user may feel relatively protected if the key space is dropped to a few hundred, as long as the login process will be locked after a few wrong tries. On the other hand, the server may not agree if the determining adversary can have a few hundred victims over a short period, because that means the adversary has a very good change of compromising at least one or two accounts.

A. *Security Analysis for Secret Little Functions*

User specified functions can be infinite. Since attackers do not know the function forms (i.e., secret encryption algorithms), these simple functions are very secure. Therefore, secret little functions can easily prevent phishing, shoulder-surfing, key-logger, and even multiple attacks. How many successful phishing attacks can be tolerated by secret little functions? The answer is infinite. For key-logger/shoulder-suffer, secret little functions can remain secure after the adversary obtains many virtual passwords.

Next, we prove that the proposed scheme can prevent the following attacks.

1) *Phishing.* Since each time, each time the user inputs a virtual password, the phishing attacker could get the virtual password, but cannot obtain the real password. The virtual password is different each time.
2) *Shoulder-surfing.* Since each time, each time the user inputs a virtual password, the shoulder-surfing attacker could get the virtual password, but cannot obtain the real password. The virtual password is different each time.
3) *Key-logger.* Since each time, each time the user inputs a virtual password, the Key-logger attacker could get the virtual password, but cannot obtain the real password. The virtual password is different each time.
4) *Replay attack.* The virtual password does not suffer the replay attack since each time, the server generated a different random number.

TABLE III
RESPONSES OF USERS

Q/A	How comfortable to do a single digit calculation?
Very easy	50%
Difficult without a calculator	2%
Ok without a calculator	19%
Easy without a calculator	29%
Q/A	Would you like to improve your password security with a little bit extra time?
Yes	41%
No	6%
Dno't care	16%
Yes, but depends on extra time	37%

We can prove mathematically that secret little function is much more secret than any other symmetric cipher (such as DES, AES, and so on) under the brute-force attack with the following theorem, while the brute-force attack is to try every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained.

Theorem 1: Under the brute-force attack, secret little function is much more secret than any other symmetric cipher.

Proof: Let P, C, and K denote the plaintext, ciphertext, and key, respectively, for any given symmetric cipher. Let L_P, L_C, and L_K denote the lengthes of P, C, and K, respectively. For an attacker to try to use the brute-force attack to the symmetric cipher, the number of alternative keys is 2^{L_K}. In other words, the attacker at worst needs to try 2^{L_K} to get the correct key. On the other hand, for a secret little function approach, the number of alternative keys is the number of secret little functions. Since the number of secret little functions is not only infinite, but also uncountable infinite, the number of alternative keys for the secret little function approach is infinite, denoted as ∞. Since we have $2^{L_K} \ll \infty$, we then prove the theorem. ■

B. *Security Analysis for Codebook and RS*

The codebook and RS approaches can prevent shoulder-surfing and key-logger since a human is not likely to tolerate more than two or three phishing attacks without being able to get into the system. RS is much stronger than the codebook approach since the function using a codebook cannot survive ten successful phishing attacks while the RS can survive an arbitrary number of successful phishing attacks.

VI. IMPLEMENTATION AND EVALUATION

We implement secret little functions and demonstrate that they defeat phishing, key-logger, and shoulder-surfing attacks in a PC machine. Even though such a calculation is complicated for some people, our helper-applications help to relieve the users of this required human computing. A user response test in the next is to test the user's feeling on the time spent to calculate the virtual password.

Our password scheme is dynamic and requires a user to make some computations. A survey in Table III is used to collect 50 users' responses for our system implementation. It was found that most of the respondents could complete the single digit calculation easily without help from the calculator. The most of the surveyed people showed their need for more secure internet with the cost of spending a little extra time.

VII. Conclusion

We discussed the challenges of protecting users' passwords on the internet and presented some related work in this field. We discussed how to prevent users' passwords from being stolen by adversaries. We proposed a virtual password concept involving a small amount of human computing to secure users' passwords in online environments. We proposed differentiated security mechanisms in which a user has the freedom to choose a virtual password scheme ranging from weak security to strong security. The function/program is used to implement the virtual password concept with a tradeoff between security and complexity and requires small amount of human computing. However, since simplicity and security conflict with each other, it is difficult to achieve both. We further proposed several functions serving as system recommended functions and provided a security analysis. We analyzed how the proposed schemes defend against phishing, key-logger, shoulder-surfing attacks, and multiple attacks. In user-specified functions, we adopted secret little functions in which security is enhanced by hiding secret functions/algorithms. In conclusion, user-defined functions (secret little functions) are better.

We believe that for some important accounts such as bank accounts, some users would like to spend a little more human computing time to make it more secure, especially when using a computer in an insure environment such as the Internet cafe.

In the future, we plan to study how to design smarter functions to alleviate the computation-burden of the user. We would also like to develop some small applications with built-in virtual passwords, which will be able to run at a customer's wireless device, such as a cellular phone or a PDA. With such an application, the user only needs to input the system random digits which the system provides and then the virtual password is automatically calculated for the user.

References

[1] T. Dierks and C. Allen, *The TLS Protocol—Version 1.0*, IETF RFC 2246, Jan. 1999.

[2] [Online]. Available: http://en.wikipedia.org/wiki/Phishing

[3] Anti-Phishing Working Group. [Online]. Available: http://www.antiphishing.org

[4] [Online]. Available: http://en.wikipedia.org/wiki/Key_logger

[5] [Online]. Available: http://www.eweek.com/article2/0,1895,1940623,00.asp

[6] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. Mitchell, "Stronger password authentication using browser extensions," in *Proc. 14th USENIX Security Symp.*

[7] E. Gaber, P. Gobbons, Y. Mattias, and A. Mayer, "How to make personalized web browsing simple, secure, and anonymous," in *Proc. Financial Crypto*, LNCS 1318. 1997.

[8] E. Gabber, P. Gibbons, D. Kristol, Y. Matias, and A. Mayer, "On secure and pseudonymous user-relationships with multiple servers," *ACM Trans. Inform. Syst. Security*, vol. 2, no. 4, pp. 390–415, 1999.

[9] E. Jung. *Passwordmaker* [Online]. Available: http://passwordmaker.mozdev.org

[10] J. la Poutr'e. *Password Composer* [Online]. Available: http://www.xs4all.nl/?jlpoutre/BoT/Javascript/PasswordComposer

[11] J. R. Levine. (2004, Apr.). *A Flexible Method to Validate SMTP Senders in DNS* [Online]. Available: http://www1.ietf.org/proceedings_new/04nov/IDs/draft-levine-fsv-01.txt

[12] V. A. Brennen. (2004). *Cryptography Dictionary*, vol. 2005, 1.0.0 ed. [Online]. Available: http://cryptnet.net/fdp/crypto/crypto-dict/en/crypto-dict.html

[13] [Online]. Available: http://www.bell-labs.com/project/lpwa

[14] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, P. Samarati, A. Tironi, and L. Zaniboni, "Spam attacks: P2P to the rescue," in *Proc. 13th Int. World Wide Web Conf.*, 2004, pp. 358–359.

[15] M. Abadi, L. Bharat, and A. Marais, "System and method for generating unique passwords," U.S. Patent 6 141 760, 1997.

[16] M. Kuhn. (1997). *Probability Theory for Pickpockets—ec-PIN Guessing* [Online]. Available: http://www.cl.cam.ac.uk/?mgk25

[17] C. Herley and D. Florencio, "How to login from an Internet cafe without worrying about keyloggers," in *Proc. SOUPS*, 2006.

[18] [Online]. Available: http://www.citibank.co.jp/en/service/cap/virtualpad

[19] [Online]. Available: http://obr.typepad.com/financial_innovations/2005/11/ing_direct_adds.html

[20] B. Moller. (1997, Feb.). *Schw¨achen des ec-PIN-Verfahrens (Manuscript)* [Online]. Available: http://www.informatik.tu-darmstadt.de/TI/Mitarbeiter/moeller

[21] A. Herzberg and A. Gbara. (2004). *Trustbar: Protecting (Even Naive) Web Users From Spoofing and Phishing Attacks*, Cryptology ePrint Archive, Rep. 2004/155 [Online]. Available: http://eprint.iacr.org/2004/155

[22] S. Wiedenbeck, J. Waters, L. Sobrado, and J. Birget, "Design and evaluation of a shoulder-surfing resistant graphical password scheme," in *Proc. Working Conf. Adv. Vis. Interfaces*.

[23] V. Roth, K. Richter, and R. Freidinger, "A PIN-entry method resilient against shoulder-surfing," in *Proc. 11th ACM Conf. Comput. Commun. Security*, 2004, pp. 236–245.

[24] IETF. (2004, Jun.). *MTA Authorization Records in DNS (MARID)* [Online]. Available: http://www.ietf.org/html.charters/OLD/marid-charter.html

[25] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," in *Proc. 12th Annu. Netw. Distributed Syst. Security Symp.*, 2005.

[26] G. T. Wilfong, "Method and apparatus for secure PIN entry," U.S. Patent #5 940 511, United States Patent and Trademark Office, Assignee: Lucent Technologies, Inc., Murray Hill, NJ, May 1997.

[27] J. Mason, "Filtering spam with SpamAssassin," in *Proc. HEANet Annu. Conf.*, 2002.

[28] D. Stinson, *Cryptography Theory and Practice*, 2nd ed. Boca Raton, FL: CRC Press, 2005.

[29] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, "A Bayesian approach to filtering junk e-mail. In learning for text categorization," in *Proc. Workshop*, May 1998.

[30] T. A. Meyer and B. Whateley, "SpamBayes: Effective open-source, Bayesian based, e-mail classification system," in *Proc. CEAS*, 2004.

[31] MAPS. (1996). *RBL—Realtime Blackhole List* [Online]. Available: http://www.mail-abuse.com/services/mds_rbl.html

[32] *The Spamhaus Project. The Spamhaus Block List* [Online]. Available: http://www.spamhaus.org/sbl

[33] Netcraft. *Anti-Phishing Toolbar* [Online]. Available: http://news.netcraft.com/archives/2004/12/28/netcraft_antiphishing_tool%bar_available_for_download.html

[34] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "SPINS: Security protocols for sensor networks," *Wirel. Netw.*, vol. 8, no. 5, pp. 521–534, 2002.

[35] Y. Xiao, C.-C. Li, M. Lei, and S. V. Vrbsky, "Secret little functions and codebook for protecting users from password theft," in *Proc. IEEE ICC*, May 2008, pp. 1525–1529.

[36] *One-Time Password* [Online]. Available: http://en.wikipedia.org/wiki/One-time_password

[37] S. Lee and K. M. Sivalingam, "An efficient one-time password authentication scheme using a smart card," *Int. J. Security Netw.*, vol. 4, no. 3, pp. 145–152, 2009.

[38] M. Abdalla, E. Bresson, O. Chevassut, B. Moller, and D. Pointcheval, "Strong password-based authentication in TLS using the three-party group DiffieHellman protocol," *Int. J. Security Netw.*, vol. 2, nos. 3–4, pp. 284–296, 2007.

[39] M. Lei, Y. Xiao, S. V. Vrbsky, and C.-C. Li, "Virtual password using random linear functions for on-line services, ATMs, and pervasive computing," *Comput. Commun. J. Elsevier*, vol. 31, no. 18, pp. 4367–4375, Dec. 2008.

[40] J. Zheng, J. Li, M. J. Lee, and M. Anshel, "A lightweight encryption and authentication scheme for wireless sensor networks," *Int. J. Security Netw.*, vol. 1, nos. 3–4, pp. 138–146, 2006.

[41] Y. Jiang, C. Lin, M. Shi, and X. Shen, "A self-encryption authentication protocol for teleconference services," *Int. J. Security Netw.*, vol. 1, nos. 3–4, pp. 198–205, 2006.

[42] J. Teo, C. Tan, and J. Ng, "Low-power authenticated group key agreement for heterogeneous wireless networks," *Int. J. Security Netw.*, vol. 1, nos. 3–4, pp. 226–236, 2006.

[43] C. Tartary and H. Wang, "Efficient multicast stream authentication for the fully adversarial network model," *Int. J. Security Netw.*, vol. 2, nos. 3–4, pp. 175–191, 2007.

[44] M. Asadpour, B. Sattarzadeh, and A. Movaghar, "Anonymous authentication protocol for GSM networks," *Int. J. Security Netw.*, vol. 3, no. 1, pp. 54–62, 2008.

[45] S. Huang and S. Shieh, "Authentication and secret search mechanisms for RFID-aware wireless sensor networks," *Int. J. Security Netw.*, vol. 5, no. 1, pp. 15–25, 2010.

[46] M. Yang, "Lightweight authentication protocol for mobile RFID networks," *Int. J. Security Netw.*, vol. 5, no. 1, pp. 53–62, 2010.

[47] J. Wang and G. L. Smith, "A cross-layer authentication design for secure video transportation in wireless sensor network," *Int. J. Security Netw.*, vol. 5, no. 1, pp. 63–76, 2010.

[48] T. Choi and H. B. Acharya, "Is that you? Authentication in a network without identities," *Int. J. Security Netw.*, vol. 6, no. 4, pp. 181–190, 2011.

[49] Q. Chai and G. Gong, "On the (in) security of two joint encryption and error correction schemes," *Int. J. Security Netw.*, vol. 6, no. 4, pp. 191–200, 2011.

[50] S. Tang and W. Li, "An epidemic model with adaptive virus spread control for wireless sensor networks," *Int. J. Security Netw.*, vol. 6, no. 4, pp. 201–210, 2011.

[51] G. Luo and K. P. Subbalakshmi, "KL-sense secure image steganography," *Int. J. Security Netw.*, vol. 6, no. 4, pp. 211–225, 2011.

[52] W. Chang, J. Wu, and C. C. Tan, "Friendship-based location privacy in mobile social networks," *Int. J. Security Netw.*, vol. 6, no. 4, pp. 226–236, 2011.

[53] X. Zhao, L. Li, and G. Xue, "Authenticating strangers in online social networks," *Int. J. Security Netw.*, vol. 6, no. 4, pp. 237–238, 2011.

[54] D. Walker and S. Latifi, "Partial Iris recognition as a viable biometric scheme," *Int. J. Security Netw.*, vol. 6, nos. 2–3, pp. 147–152, 2011.

[55] A. Desoky, "Edustega: An education-centric steganography methodology," *Int. J. Security Netw.*, vol. 6, nos. 2–3, pp. 153–173, 2011.

[56] N. Ampah, C. Akujuobi, S. Alam, and M. Sadiku, "An intrusion detection technique based on continuous binary communication channels," *Int. J. Security Netw.*, vol. 6, nos. 2–3, pp. 174–180, 2011.

[57] H. Chen and B. Sun, "Editorial," *Int. J. Security Netw.*, vol. 6, nos. 2–3, pp. 65–66, 2011.

[58] M. Barua, X. Liang, R. Lu, and X. Shen, "ESPAC: Enabling security and patient-centric access control for eHealth in cloud computing," *Int. J. Security Netw.*, vol. 6, nos. 2–3, pp. 67–76, 2011.

[59] N. Jaggi, U. M. Reddy, and R. Bagai, "A three dimensional sender anonymity metric," *Int. J. Security Netw.*, vol. 6, nos. 2–3, pp. 77–89, 2011.

[60] M. J. Sharma and V. C. M. Leung, "Improved IP multimedia subsystem authentication mechanism for 3G-WLAN networks," *Int. J. Security Netw.*, vol. 6, nos. 2–3, pp. 90–100, 2011.

[61] N. Cheng, K. Govindan, and P. Mohapatra, "Rendezvous based trust propagation to enhance distributed network security," *Int. J. Security Netw.*, vol. 6, nos. 2–3, pp. 101–111, 2011.

[62] A. Fathy, T. ElBatt, and M. Youssef, "A source authentication scheme using network coding," *Int. J. Security Netw.*, vol. 6, nos. 2–3, pp. 112–122, 2011.

[63] L. Liu, Y. Xiao, J. Zhang, A. Faulkner, and K. Weber, "Hidden information in microsoft word," *Int. J. Security Netw.*, vol. 6, nos. 2–3, pp. 123–135, 2011.

[64] S. S. M. Chow and S. Yiu, "Exclusion-intersection encryption," *Int. J. Security Netw.*, vol. 6, nos. 2–3, pp. 136–146, 2011.

[65] X. Lin, X. Ling, H. Zhu, P. Ho, and X. Shen, "A novel localized authentication scheme in IEEE 802.11 based wireless mesh networks," *Int. J. Security Netw.*, vol. 3, no. 2, pp. 122–132, 2008.

[66] A. Scannell, A. Varshavsky, A. LaMarca, and E. De Lara, "Proximity-based authentication of mobile devices," *Int. J. Security Netw.*, vol. 4, nos. 1–2, pp. 4–16, 2009.

[67] J. M. McCune, A. Perrig, and M. K. Reiter, "Seeing-is-believing: Using camera phones for human-verifiable authentication," *Int. J. Security Netw.*, vol. 4, nos. 1–2, pp. 43–56, 2009.

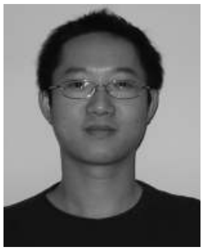
[68] S. Laur and S. Pasini, "User-aided data authentication," *Int. J. Security Netw.*, vol. 4, nos. 1–2, pp. 69–86, 2009.

Yang Xiao (SM'04) was with the industry as a Medium Access Control Architect involving the IEEE 802.11 standard enhancement work before he joined the Department of Computer Science, University of Memphis, Memphis, TN, in 2002. He is currently with the Department of Computer Science (with tenure), University of Alabama, Tuscaloosa. His research has been supported by the U.S. National Science Foundation (NSF), U.S. Army Research, The Global Environment for Network Innovations, Fleet Industrial Supply Center-San Diego, FIAT-ECH, and the University of Alabama's Research Grants Committee. His current research interests include security and communications/networks. He has published more than 200 refereed journal papers (including 50 IEEE/ACM transaction papers) and over 200 refereed conference papers and book chapters related to these research areas.

He was a Voting Member of the IEEE 802.11 Working Group from 2001 to 2004. He serves as a panelist for the U.S. NSF, Canada Foundation for Innovation's Telecommunications Expert Committee, and the American Institute of Biological Sciences, as well as a referee/reviewer for many national and international funding agencies. He currently serves as the Editor-in-Chief for the *International Journal of Security and Networks* and the *International Journal of Sensor Networks*. He was the Founding Editor-in-Chief for the *International Journal of Telemedicine and Applications* from 2007 to 2009.

Chung-Chih Li received the Ph.D. degree in computer and information science from Syracuse University, Syracuse, NY.

He is currently an Associate Professor with the School of Information Technology, Illinois State University, Normal. He teaches several programming courses at different levels and coaches his students in ACM programming contests. His primary research interests include theoretical computer science. He is particularly interested in higher ordered computation. He has extended his research interests into cryptography, security, WSN, and computational learning theory.



Ming Lei received the B.A. degree from Southwest Jiaotong University, Chengdu, Sichuan, China, in 2000, and the Ph.D. degree in computer science from the University of Alabama, Birmingham, in 2008.

He is currently a Principal Software Engineer with Oracle Corporation, Redwood City, CA, in which, as a Technical Leader, he designs and develops advanced analytical applications for retailers to optimize and plan their pricing strategy. His current research interests include real-time database systems, networking security, high-performance grid computing, mobile data management, and data mining.

Susan V. Vrbsky received the B.A. degree from Northwestern University, Evanston, IL, the M.S. degree in computer science from Southern Illinois University, Carbondale, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, Urbana.

She is currently an Associate Professor of computer science with the Department of Computer Science, University of Alabama, Tuscaloosa. Her current research interests in database systems include cloud computing, data grids, green computing, real-time database systems, and database security.