Public-key cryptography and password protocols

Shai Halevi* Hugo Krawczyk†

Abstract

We study protocols for strong authentication and key exchange in asymmetric scenarios where the authentication server possesses a pair of private and public keys while the client has only a weak human-memorizable password as its authentication key. We present and analyze several simple password protocols in this scenario, and show that under the choice of suitable public key encryption functions the security of these protocols can be formally proven based on standard cryptographic assumptions. In particular, our analysis shows optimal resistance to off-line password guessing attacks. In addition to user authentication, we enhance our protocols to provide two-way authentication, authenticated key exchange, defense against server's compromise, and user anonymity. We complement these results with a proof that public key techniques are unavoidable for password protocols that resist off-line guessing attacks.

As a further contribution, we introduce the notion of *public passwords* that allow for the use of such protocols in situations where the client's machine does not have the means to validate the server's public key. Public passwords serve as "hand-held certificates" that the user can carry without the need for special computing devices.

1 Introduction

In this paper we study the use of human passwords for strong authentication and key exchange in *asymmetric* scenarios where the authentication server can store a strong secret (such as the private key for public-key encryption) while the client uses a weak human-memorizable password as its only authentication key. This asymmetry arises naturally in applications, such as remote user authentication, where the user does not carry any computational device (e.g., a laptop or smartcard) capable of storing a long secret. It also arises in applications of protocols such as SSL, IPSEC and SET where the client end does not possess a public key[1].

The first work to deal with the use of public key techniques in conjunction with password authentication was by Gong, Lomas, Needham and Saltzer [11]. In that paper, it was suggested that by providing the authentication server with a pair of private/public keys one could protect weak human passwords against strong attacks via the use of public key encryption.

The emphasis of that work was in protecting passwords against off-line password-guessing attacks, which is a common and powerful attack that takes advantage of the low entropy available in user-chosen passwords. This attack proceeds by trying to guess the password and verify the guessed value using publicly available information, such as the transcript of a legitimate authentication session between the user and server. This attack is very powerful, since it can be performed off-line, so the attacker does not need to interact with the legitimate parties, and can use a lot of computing power. (In contrast, on-line attacks where the attacker actively tries different passwords against the server are easy to detect and limit.) Our work extends the public key based approach of [11] in several ways.

- We consider several simple and intuitive protocols for password authentication, and formally analyze their security. We show that the security of these protocols strongly depends on the choice of the public key encryption function, and demonstrate how some natural and "seemingly secure" realizations of the protocols can be broken. On the other hand, by strengthening the notion of encryption (to resist some form of chosen ciphertext attack) we can formally prove the security of the protocols. In particular, we show optimal resistance to off-line password-guessing attacks.
- We enhance our basic protocols to provide important features such as two-way authentication, authenticated key exchange, resistance to server's compromise, and user anonymity.
- We introduce the notion of *public passwords*, as an enabler for the use of our protocols in situations where the certified server's public key is not available to the client's machine.
- Finally, we prove a general theoretic result, showing that the use of public key techniques is unavoidable in password protocols that provide defense against off line guessing attacks.

* IBM T.J.Watson Research Center, PO Box 704, Yorktown Heights, New York 10598, USA Email: shaih@watson.ibm.com.

† Department of Electrical Engineering, Technion, Haifa 32000, Israel, and IBM T.J. Watson Research Center, New York, USA. Email: hugo@ee.technion.ac.il.

[1] SET defines a "certless" option to support these cases. SSL and IPSEC currently do not define such modes of authentication but the need for them has been repeatedly pointed out.

Security of password protocols. The main difficulty in designing secure password mechanisms arises from the fact that the space of passwords is usually small and much easier to attack than random cryptographic keys. In particular, exhaustive search attacks as the off-line guessing attacks mentioned above become practical. Moreover, using a low-entropy password as a key to a cryptographic function, can transform an otherwise strong function into a weak one. Namely, when using passwords as cryptographic keys, one makes the assumption that these functions remain secure even when the keys are chosen from a very small set. These assumptions are so unusual that, to the best of our knowledge, no one has been able to formally define the requirements from these cryptographic functions under which existing protocols can be proved secure.[2]

Our work avoids these problems by providing mechanisms that do not use the password as a key to cryptographic functions and which we formally prove secure based on standard cryptographic assumptions. Our assurance of security is very strong: we show that the attacker cannot do better than just trying its luck in active (on-line) impersonation attempts (e.g., by trying to authenticate to the server using a guessed password). Note that if the attacker performs v such attempts and the password is taken from a dictionary of size d then the attack can succeed with probability v/d. What we show is that additional off-line work by the attacker does not help improving significantly this probability (as long as the attacker cannot break the encryption function).

Security of public key encryption. In this work we use public key encryption to design password authentication protocols. Although the protocols themselves are very simple and intuitive, proving their security is not straightforward. In particular, it turns out that the basic notion of security for encryption algorithms that guarantees secrecy against eavesdroppers is not sufficient to ensure the security of these protocols, and that a stronger notion has to be used. We briefly discuss these notions here.

The basic notion of security for public key encryption, called *semantic security*, was introduced by Goldwasser and Micali [10]. In a nutshell, an encryption scheme is said to be semantically secure if, given a ciphertext c and a plaintext p, it is infeasible to determine whether or not c is an encryption of p. (Clearly, such an encryption algorithm must be randomized, so that simply re-encrypting p and comparing the result to c does not work.) This, in turn, implies other strong properties of the encryption function such as the infeasibility to derive any partial information on the encrypted plaintext given its ciphertext.

Although semantic security provides very high level of secrecy protection, it is still not enough to ensure the security of our protocols. Indeed, in Section 3.6 we show how the use of a particular semantically secure encryption function leads to an *insecure* implementation of the protocols (we show a successful off-line guessing attack against such implementation). Hence a stronger notion of security of the encryption algorithm is needed. This stronger notion of security, known as resistance to *chosen ciphertext attacks*, was introduced by Rackoff and Simon [21]. It requires that an adversary cannot determine whether or not c is an encryption of p, even when it is given some "extra help" in the form of the ability to ask for the decryption of ciphertexts of its choice (but not for the decryption of c).

In fact, in this work we use a seemingly weaker form of chosen ciphertext attacks, which we call *one-ciphertext verification* attacks. In this form, the "extra help" that the adversary gets is limited to the ability to generate a single pair (p', c') of plaintext and ciphertext (with $c' \neq c$), and to ask whether or not c' is an encryption of p'. Namely, we make the assumption that even with this extra help, an adversary cannot determine whether or not c is an encryption of p, and use this assumption on the encryption algorithm to prove that our password protocols are secure.

Public passwords. Our protocols enjoy several attractive properties and are suited for implementation in cases where the authentication server possesses a public key. However, this requires the client machine to know the *correct* value of this public key. Under certain circumstances this is possible via a certification of the server's public key by a trusted party or via some other form of trusted distribution to the client machine. However, if a user needs to authenticate from a remote machine that does not have a way to validate the correct public key of the authentication server, then the security of these protocols is in danger. In these cases, we propose to provide the users with a *digest* of the server's public key. This digest, typically of length 60-80 bits, does not need to be memorized by the user. It can be safely written on paper, a plastic card, etc. In Section 5 we discuss several ways to implement these digests; we also show that the user does not even need to type such a digest, but just recognize it when displayed. We call this digest a *public password*. This notion may be of significant practical value beyond our applications in order to bootstrap trust in public keys before a public key infrastructure is in place (and possibly even after that).

Necessity of public key techniques. All the published solutions for password authentication that resist guessing attacks use public key techniques; we show that this is no accident. Specifically, we prove that every authentication protocol which resists off-line password-guessing attack requires the use public key techniques. More precisely, we show that given any password protocol resistant to off-line guessing attacks one can build a secure key-exchange protocol. Using a result from [13], our proof implies that a secure password authentication protocol cannot be devised using only simple symmetric cryptographic primitives such as hash functions, block ciphers or pseudorandom functions (barring some major breakthrough in cryptographic and complexity-theoretic research). This provides yet another indication for the inherent complexity of the design of secure password protocols.

Related work. We have already mentioned the work of Gong et al. [11] which was also the first to deal with the problem of guessing attacks against password protocols. Another very influential work, by Bellovin and Merrit [4], introduced Encrypted Key Exchange (EKE) which became the basis for many of the subsequent works in this area, e.g. [5, 14, 22, 18, 20, 23]. The security of these solutions is based on heuristic assumptions about the underlying cryptographic functions. Other forms of password authentication, such as one-time passwords, are discussed in Section 2. For a survey of works and techniques related to password authentication see [19, 15].

[2] It may be possible to define these requirements in terms of idealized assumptions like the random oracle model; see [18]. However, when we replace these ideal functions with actual cryptographic functions the security of the resultant scheme is unknown.

Organization. In Section 2 we briefly discuss basic password mechanisms and the security requirements for password-based authentication and key exchange. In Section 3 we present our protocols and their security analysis. In Section 4 we prove the necessity of public key techniques for designing password authentication resistant to password-guessing attacks. Finally, in Section 5 we expand on the notion of public passwords and discuss some issues regarding their implementation.

2 Password Mechanisms and Their Security

In this section we first briefly discuss several mechanisms for password authentication, and then describe formally the notion of security for these mechanisms which we use in this paper. A reader who is familiar with password security issues can skip most of this section and go right to our security definition in Subsection 2.4.

2.1 Basic password mechanisms

Password transmission. The simplest password mechanism is the transmission of a password in the clear from the user to the server. To validate the password, the server stores a file containing either the plain passwords (attached to the user name) or an image of the passwords under a one-way function. The latter is the classic method of the `Unix` system and is used in remote authentication for functions like `ftp` and `telnet`. In the case of remote authentication the drawback of this mechanism is clear as the password can be easily read by an eavesdropper from the network.

Challenge response. A more secure form of password authentication uses the so called challenge response mechanisms. In this case the password is never transmitted in the clear but is used to compute a secret function on a challenge which is selected by the authentication server with each new authentication instance. This provides freshness for the authentication but leaves the password open to *password-guessing attacks* which work as follows. The attacker is assumed to have access to a relatively small dictionary, containing many common passwords. It first records an authentication session including the challenge and the corresponding response from the user. Later, the attacker tries a set of possible passwords on the challenge to see whether the same response is obtained. If so, the password was found (with high probability). Unfortunately, in reality many passwords are indeed found in such dictionaries, thus the above attack is highly effective.

One-time passwords. One variant of challenge-response mechanisms is the so called "one-time password" authentication (e.g. see [17, 12]), in which the user uses a different password every time it tries to authenticate itself. If these one-time passwords are derived from a human password, the latter is still vulnerable to password-guessing attacks. This can be avoided by providing the user with a list of one-time passwords written on paper. This has the advantage that such a password cannot be re-used (even by the local terminal or by somebody breaking into the authentication server). However it entails the inconvenience for the user of carrying a long list of passwords, the requirement to keep this list safe and secret, and the need to type relatively complex strings into the terminal. In addition, this mechanism is vulnerable to several attacks, ranging from stolen passwords (e.g., copied from the person's paper) to man-in-the-middle attacks, and does not support the important extensions discussed in the next subsection.

2.2 Beyond simple authentication

Password mechanisms can provide additional functionalities on top of one-way authentication of user to server. In particular, we often need them to have the following features.

- *Mutual authentication.* Not only the user authenticates itself to the server but the server authenticates to the user as well. This is important to avoid man-in-the-middle and server impersonation attacks. The importance of such two-way authentication increases with the need to authenticate remote users over completely untrusted networks like the Internet.
- *Authenticated key-exchange.* At the end of the protocol, the user and the server share a secret session key. This session key is then used to authenticate or encrypt subsequent communication in the current session. (This prevents hijacking of sessions by an intruder, data forgery and data exposure.)
- *User identity protection.* An eavesdropper to the authentication protocol does not learn the identity of the user. (This is particularly important with remote authentication of mobile users.)

In all these cases the strength of the authentication provided by the password mechanism is usually the security bottleneck for the added functionalities.

2.3 Security of password authentication

Below we describe the notion of security for password authentication which we use in this work. We first present a list of basic attacks that one needs to guard against, and then provide a more formal definition.

- *Eavesdropping.* The attacker listens on the line and tries to learn some useful information from the ongoing communication.
- *Replay.* The attacker records messages which were sent in past communications and re-sends them at a later time.
- *Man-in-the-middle.* The attacker intercepts the messages sent between the parties and replaces them with its own messages. It plays the role of the user in the messages which it sends to the server, and at the same time plays the role of the server in the messages that it sends to the user.
- *Password-Guessing attacks.* The attacker is assumed to have access to a relatively small *dictionary* containing common choices of passwords. There are primarily two ways in which the attacker can use the dictionary
  - *Off-line attack.* The adversary records past communication, and then goes over the dictionary and looks for a password which is consistent with the recorded communication. If such a password is found, the attacker concludes that this is the password of the user.
  - *On-line attack.* The attacker repeatedly picks a password from the dictionary and tries to use it in order to impersonate as the user. If the impersonation fails, the attacker eliminates this password from the dictionary and tries again, using a different password. The standard ways of preventing

such on-line attacks in practice are to either limit the number of failed runs that a user is allowed to have before the password is expired, or reduce the rate in which the user is allowed to make login attempts.

In addition to the attacks above, an attacker may occasionally also get access to sensitive data which is supposed to be kept secret at the participating parties. In this case the goal is to minimize the effect that the compromise of any single key or file has on the entire system. (An example for this is the notion of perfect forward secrecy.) In particular, in the context of passwords mechanism, one needs to consider the effect of a compromised password on the derived session-keys (and vice-versa), and the effects of compromising the password file or the secret-key of the server. In principle, the compromise of any of these secrets can potentially influence any of the other secrets.

2.4 Definition of basic security

Here we sketch our definition for the most basic notion of security for a password-based one-way authentication protocol. (This definition includes resistance to the above attacks, but says nothing about the effects of key-compromise.) We present the notion of security through the description of the attacker that we consider. The attacker, which we call a *forger* F, is allowed to watch regular runs of the protocol between the *user* U and the *server* S, and can also actively communicate with the user and the server in replay, impersonation and man-in-the middle attacks. The forger can initiate multiple authentication sessions with S as if they were requested by U. It can intercept challenges sent from S to U and change them to any value of its choice, and then see the response from U to that (possibly modified) challenge. The forger can also intercept responses from U to S and change their values and then send the (possibly modified) response to S. Finally, F also gets to see whether S accepts the authentication or not. We say that the forger breaks the authentication protocol if S accepts a response which was sent to it by F but was not freshly generated by U.

It is clear that if the user picks its password from a dictionary $\mathcal{D}$, then a forger that attempts v active impersonation attacks with the server can break the protocol with probability of at least $v/|\mathcal{D}|$ (just by trying in each impersonation attempt a different password from $\mathcal{D}$).[3] A protocol is considered secure if F cannot do significantly better than this trivial bound.

Definition 1 *Let ϵ be any positive real number. We say that a one-way password authentication protocol enjoys basic security up to ϵ, if no feasible*[4] *attacker F can break the protocol with probability higher than $\frac{v}{|\mathcal{D}|} + \epsilon$, using only v impersonation attempts with the server.*

[3] For simplicity we assume that users choose passwords from $\mathcal{D}$ uniformly at random; however, our analysis can easily accommodate any other distribution.

[4] The computational "feasibility" of attackers can be formalized as polynomial-time or, concretely, using some specific computational bounds. In particular, we want to consider realistic attackers that can perform an exhaustive search over a password dictionary but cannot break the cryptographic primitives used in the protocol. For example, it is reasonable to assume that an attacker can carry out 2^{40} computational steps, that may be enough to exhaust a password dictionary, but not 2^{80} as may be needed to break some encryption scheme.

We note that keeping the value $v/|\mathcal{D}|$ small should be enforced by auditing mechanisms that either expire a password after a certain number of failed authentication attempts, or limit the rate in which such attempts can be made.

3 Password Authentication Using the Server's Public Key

Below we present several protocols in the asymmetric scenario where the authentication server is assumed to have a pair of private and public keys while the client authenticates on the basis of a user's password. It is a requirement for the security of these protocols that the server's public key used by the client for encryption be a valid public key for that server. As discussed in the Introduction (see also Section 5) when the client machine does not have a way to validate this public key we suggest to provide the user with a *public password* that acts as a "hand-held certificate" for that key. For the sake of illustration and concreteness, we assume in the description of our protocols the availability of such a public password.

For clarity of presentation and analysis we start by presenting only the basic protocols for user authentication. Later (Section 3.4) we augment these protocols to provide mutual authentication and key exchange. We first introduce some terminology and tools common to our different protocols.

Terminology and tools: The user name is denoted by U and the server's name is denoted by S. A flow of a protocol is denoted with arrows. For example, $S \leftarrow m \leftarrow U$ means message m sent from U to S. The secret password memorized by the user U is denoted by spwd (typically, the value of spwd is computed as a hash value of the user's typed secret password). The public key of the server S is denoted pk_S. The public password of the user is denoted by ppwd. In all the protocols of this section we have $\mathsf{ppwd} = MD(\mathsf{pk}_S)$, where MD is a collision-resistant (or second pre-image resistant) hash function, e.g SHA-1.

In the protocols below the symbol r stands for a "challenge", namely, a random string freshly chosen by S with each protocol execution and long enough as to have only negligible probability of ever repeating. The symbol ENC stands for a randomized encryption scheme, which we assume is resistant to some forms of chosen ciphertext attack (see below). We denote encryption using ENC under the public key of S by $\text{ENC}_{\mathsf{pk}_S}$.

In the protocols of Section 3.4 we use families of pseudorandom functions [9] which we denote by PRF. An individual function in the family is indexed by its key, e.g. PRF_k. We use pseudorandom functions for key derivation as well as message authentication codes. Typical implementations of pseudorandom functions are based on block ciphers and cryptographic keyed hash functions.

In our description of protocols we omit an initialization flow in which U communicates to S the fact that he wants to talk to S. Also, for simplicity, we omit explicit specification of some of the obvious (yet essential) verification steps by the parties.

3.1 Encrypted password transmission

We start by presenting an extremely simple protocol where the password is encrypted with the server's public key, and then sent to the server for verification. We note that this protocol is reminiscent of the Identification Protocol of [11] (but is even simpler). Later we augment this protocol to provide stronger security properties.

Encrypted Password Transmission

Set-up: $\mathsf{ppwd} := MD(\mathsf{pk}_S)$		
S		U
Pick a nonce r	$\rightarrow\ \mathsf{r}, \mathsf{pk}_S\ \rightarrow$	
		Check $\mathsf{ppwd} = MD(\mathsf{pk}_S)$
Verify password	$\leftarrow\ \textsc{Enc}_{\mathsf{pk}_S}(\mathsf{r}, \mathsf{spwd})\ \leftarrow$	

Notice that the public password ppwd identifies the public key as being the authentic server's public key, and thus the user can safely use it to encrypt its password. We remark that this simple protocol does not rely on the password as a cryptographic key by itself, thus it avoids the weakness of choosing a cryptographic key from a too small space as discussed earlier in this paper. (We also stress again that the encryption scheme is *randomized*, and thus an attacker cannot cannot simply guess spwd and verify the guess using $\textsc{Enc}_{\mathsf{pk}_S}(\mathsf{r}, \mathsf{spwd})$.)

An important aspect of this protocol is the use of the *challenge* r sent from S to U. This acts as a proof of freshness without which the authentication could be trivially broken by replaying the ciphertext. However, this protocol uses the encryption function Enc not only to hide the password but also to *bind* the challenge to the password. Thus, it requires that the encryption function will have other properties than simply hiding the encrypted message. For example, it should be infeasible (without knowing spwd) to obtain $\textsc{Enc}_{\mathsf{pk}_S}(\mathsf{r}', \mathsf{spwd})$ from $\textsc{Enc}_{\mathsf{pk}_S}(\mathsf{r}, \mathsf{spwd})$ for some other $\mathsf{r}' \neq \mathsf{r}$. For example, even using a perfect one-time pad encryption of $(\mathsf{r}, \mathsf{spwd})$ would be insecure here since modifying it to an encryption of $(\mathsf{r}', \mathsf{spwd})$ is trivial even without knowing the encryption key. Similarly, ElGamal encryption is vulnerable to such an attack, too. Our analysis below provides a characterization of the additional properties required from the encryption function in order to ensure the security of this protocol.

The above simple protocol is a special case of a broader (and more powerful) family of protocols that we call "encrypted challenge-response" mechanisms and that we describe and analyze next.

3.2 Generic encrypted challenge-response protocol

Here we propose to use the challenge-response approach but to encrypt the user's response under the server's public key as a means to protect against password-guessing attacks. It turns out that this approach, although natural and intuitive, is less straightforward than it may seem at first glance. Indeed, the intuition that hiding the response from an attacker should be enough to prevent guessing-attacks is false. To stress this point we present in Section 3.6 a password-guessing attack against a particular "bad implementation" of this protocol, that succeeds even when the public key encryption scheme in use provides provable secrecy protection against eavesdroppers. Fortunately, we are able to show that under a stronger (yet achievable) notion of security for the encryption function, our protocols are provably secure.

Another drawback of regular challenge-mechanisms that we need to eliminate is the use of weak human passwords as keys to cryptographic functions; we note that the security of these functions under such a small space of keys is clearly questionable. In contrast, in our basic authentication protocols we use the password under functions that do not have any "cryptographic requirements" related to the password but just require very simple combinatorial properties (e.g., being one-to-one).

We first present a skeleton protocol using a *generic challenge-response* function f that combines the password spwd and challenge r into some response. Next we analyze the security of this protocol in terms of the structure of f and the security of the encryption function. Armed with this information we then proceed to suggest concrete protocols that achieve secure user authentication.

Generic Encrypted Challenge-Response Protocol

Set-up: $\mathsf{ppwd} := MD(\mathsf{pk}_S)$		
S		U
Pick a nonce r	$\rightarrow\ \mathsf{r}, \mathsf{pk}_S\ \rightarrow$	
		Check $\mathsf{ppwd} = MD(\mathsf{pk}_S)$
Decrypt and verify	$\leftarrow\ \textsc{Enc}_{\mathsf{pk}_S}(f(\mathsf{spwd}\,;\mathsf{r}, U, S))\ \leftarrow$	

Security of the encryption function Enc. For the security of the above protocol, we must assume that the encryption function Enc resists a weak form of chosen ciphertext attacks, which we call *one-ciphertext verification attacks*: Let $\mathcal{E} = (\textsc{Gen}, \textsc{Enc}, \textsc{Dec})$ be an encryption scheme, where Gen is the key-generation algorithm, Enc is the (probabilistic) encryption algorithm and Dec is the decryption algorithm. A one-ciphertext verification attack is formally defined via the following experiment, which involves the three algorithms and an adversary A.

1. The key-generation algorithm is run (with security parameter k), to generate a secret/public key pair, $(\mathsf{sk}, \mathsf{pk})$.
2. The adversary A is given pk, and it generates a message x.
3. With probability 1/2, the adversary gets $c = \textsc{Enc}_{\mathsf{pk}}(x)$. Otherwise, a string s of the same length as x is chosen uniformly at random, and the adversary gets $c = \textsc{Enc}_{\mathsf{pk}}(s)$.
4. The adversary generates a query (x', c') with $c' \neq c$, and it is told whether or not $x' = \textsc{Dec}_{\mathsf{sk}}(c')$.
5. The adversary guesses whether or not $x = \textsc{Dec}_{\mathsf{sk}}(c)$.

Definition 2 *An encryption scheme $\mathcal{E} = (\textsc{Gen}, \textsc{Enc}, \textsc{Dec})$ is said to* resist one-ciphertext verification attacks *with security $\epsilon = \epsilon(k)$ if for any polynomial time A*

$$\left|\ \Pr[A \textit{ guesses ``encryption of } x\textit{''} \mid c = \textsc{Enc}_{\mathsf{pk}}(x)] - \Pr[A \textit{ guesses ``encryption of } x\textit{''} \mid c = \textsc{Enc}_{\mathsf{pk}}(s)]\ \right| \leq \epsilon$$

where the probabilities are taken over the execution of Gen, *the coins of A, and the randomness used in Step 3 above.*

Structure of the function f. Below we assume that the function $f(\cdot\,;\cdot)$ has the property that for every fixed strings spwd, x, the induced functions $f(\mathsf{spwd};\cdot), f(\cdot\,;x)$ are one-to-one. We say that a function f as above is *one-to-one on its components*. (For example, the concatenation function $f(x;y) = (x,y)$ has this property, as does the XOR function if x and y are of the same length.) We note that in fact, it is sufficient that $f(\mathsf{spwd};\cdot)$ be collision-resistant and it does not actually have to be one-to-one. Nonetheless, for clarity of presentation we assume below that it is one-to-one.

Theorem 1 *Let ϵ be any positive real number, let $\mathcal{E}$ be an encryption scheme that resists one-ciphertext verification attacks with security ϵ, and let f be one-to-one on its components. Then the encrypted challenge-response protocol using $\mathcal{E}$ and f enjoys basic security up to $\epsilon' = M \cdot \epsilon$, where M is*

at most quadratic in the number of messages that are sent by the attacker during active impersonation attacks against the protocol.[5]

We present the proof of Theorem 1 in Section 3.5.

Corollary 1 *The encrypted password transmission protocol is secure under the above assumptions on the encryption scheme $\mathcal{E}$.*

Proof. Obviously the concatenation function $f(\mathsf{spwd}\,;\mathsf{r},U,S) = (\mathsf{spwd},\mathsf{r},U,S)$ is one-to-one on its components. ■

Remark (*chosen ciphertext security*). Stronger notions of security against chosen ciphertext attacks can be found in [21, 7, 3, 2]. In particular, Bellare and Rogaway presented in [3] a simple encoding of data (called OAEP) for use with RSA encryption that provides defense against strong attacks. Although the analysis of that construction is given on the basis of ideal random functions, it should be considered as a good heuristic and, in particular, advisable for use in our (less demanding) scenario. Also, very recently Cramer and Shoup [6] described a simple encryption scheme which is provable secure against the strongest type of chosen ciphertext attacks without using an "ideal random function".

Remark (*user anonymity*). It is possible to derive, from the above generic scheme, protocols in which the user's identity is only sent encrypted in the second flow under the server's public key. In this case, the remote terminal first informs the server of a request for authentication but does not specify the user. In the above protocol, S can send ppwd and the challenge without knowing the identity of the specific user (here we use the fact that all users carry the same value of ppwd). This provides the important *user anonymity* property in cases of remote and mobile authentication.

3.3 Resistance to server compromise

Although the above proof implies that every implementation of the encrypted challenge response protocol enjoys basic security, different implementations may have very different security properties with respect to the compromise of the secret information stored on the server. (For example, it is clear that the encrypted password transmission protocol of Section 3.1 becomes totally insecure once the private key of the server is compromised).

To protect against compromise of the server, one can use some common heuristics for the definition of the f function. For example, one can set

$$\begin{aligned} p_1 &= H_1(\mathsf{spwd},U,S)\\ p_2 &= H_2(\mathsf{spwd},U,S)\\ p_3 &= H_3(p_2,\mathsf{salt})\\ f(\mathsf{spwd};\mathsf{r},U,S) &\stackrel{\text{def}}{=} \langle \textsc{Mac}_{p_1}(\mathsf{r},U,S),p_2,\mathsf{r}\rangle \end{aligned}$$

and have the server store the values salt, p_3 and p_1 (where H_1, H_2, H_3 are one-way functions, MAC is a message authentication code, and salt is a random string).

The above mechanism defends against compromise of *either* password file or server's private key (but not simultaneously against both[6]). If the password file is compromised but the server's private key remains secret, then the attacker still needs to mount a password-guessing attack to find p_2. If, on the other hand, the attacker gets the server's private key but does not gain access to the password file, then it still cannot trivially authenticate the user since it needs to be able to compute the value $\textsc{Mac}_{p_1}(\mathsf{r},U,S)$.

We stress that in this case we are making the heuristic assumption that the attacker cannot break the MAC function in any better way than a password-guessing attack. As said before this is a non-standard assumption for most MAC functions since they were designed to be keyed over a much larger space. Still, in our case this assumption is not the basis for the authentication security but only a second line of defense in case of server's key compromise.

3.4 Mutual authentication and key exchange

Here we add to the above basic authentication protocols the capability of authenticating the server to the user as well as of exchanging an authenticated secret key between the two. This added functionality is needed in many security applications. In particular, our solutions can provide authenticated key exchange for protocols such as IPSEC and SSL where the client's end authenticates via a user's password.

Our extensions for mutual authentication and key exchange follow the general design of SKEME [16]. The basic idea is that the user U adds an encryption of a (random) key k to the authentication information that it sends to S in the second flow of the protocol. The server S uses this key to authenticate itself, by using k as a key to a MAC function. (It is the sole ability of S to decrypt k which forms the basis for the server's authentication.) A shared key can be derived by applying a pseudorandom function, keyed with the above key k, to the exchanged information. (In actuality, we use the pseudorandom function PRF in this protocol both as a MAC and for key derivation.) Note that here the strength of the authentication from S to U is based on the cryptographic keys of S and hence it is stronger than in other mechanisms that base this authentication on the strength of the user's password. An analysis of this later form of authentication based on public key encryption resistant to chosen ciphertext attacks can be found in [1].

Mutual Authentication and Key Exchange

	Set-up: $\mathsf{ppwd} := MD(\mathsf{pk}_S)$	
S		U
Pick a nonce r	$\rightarrow\ \mathsf{r},\mathsf{pk}_S\ \rightarrow$	Check $\mathsf{ppwd} = MD(\mathsf{pk}_S)$
		Pick a random key k
	$\leftarrow\ \textsc{Enc}_{\mathsf{pk}_S}(k, f(\mathsf{spwd}\,;\mathsf{r},k,U,S))\ \leftarrow$	
Decrypt and verify		
Set $y := \textsc{prf}_k(\mathsf{r},S,U)$	$\rightarrow\ y\ \rightarrow$	Check $y = \textsc{prf}_k(\mathsf{r},S,U)$
Set $k' := \textsc{prf}_k(y)$		Set $k' := \textsc{prf}_k(y)$

Note that the two first flows are the same as in the generic encrypted challenge response protocol of section 3.2 except that the key k is included in the encryption and in the function f.

The above protocol does not provide perfect forward secrecy, since if the servers private key is eventually exposed

[5] For example, if ENC has security $\epsilon = 2^{-80}$, and the password expires after 100 failed trials, then we get $\epsilon' = 2^{-80}\cdot 100^2 \approx 2^{-67}$. Thus, any attacker has probability of at most $100/|\mathcal{D}| + 2^{-67}$ for a successful impersonation.

[6] Defense against compromise of both the password file and the server's private key can be achieved by using Lamport's one-time password mechanism [17, 19] instead of the fixed value p_2. This mechanism requires a pre-established limit on the number of password authentications before the password value in the server is to be re-set.

then the session key k' is revealed. As with any key-exchange protocol, perfect forward secrecy can be added through the use of Diffie-Hellman exchange. The resulting protocol is as follows (below we assume a common prime modulus over which the DH exchange is carried. We omit the mod p notation.)

Mutual Authentication and Diffie-Hellman Key Exchange

Set-up: $\mathsf{ppwd} := MD(\mathsf{pk}_S)$		
S		U
Pick r, g^x	$\rightarrow\ \mathsf{r}, g^x, \mathsf{pk}_S\ \rightarrow$	Check $\mathsf{ppwd} = MD(\mathsf{pk}_S)$
		Pick at random k, g^y
		$t := f(\mathsf{spwd}\,;\mathsf{r}, g^x, g^y, k, U, S)$
Decrypt and verify	$\leftarrow\ g^y, c\ \leftarrow$	$c := \text{ENC}_{\mathsf{pk}_S}(k, t)$
Set $z := \text{PRF}_k(c)$	$\rightarrow\ z\ \rightarrow$	Check $z = \text{PRF}_k(c)$
Set $k' := \text{PRF}_k(g^{xy})$		Set $k' := \text{PRF}_k(g^{xy})$

We note that is g^x is chosen at random in every run, then the challenge r is not needed and can be omitted. The derivation of the session key through the application of PRF_k to the DH key g^{xy} is intended to "hash" the DH key into a shorter and stronger key (it also makes the protocol resistant to the breaking of either the Diffie-Hellman exchange or the encryption function, namely, to compute k' an attacker needs to be able to compute g^{xy} and also to find the value k'). Finally, we stress that the information in the second argument of the function f can be hashed under a collision resistant hash function (such as SHA-1) before computing f on it. This preserves the security properties that we prove and shortens the information to fit under the encryption.

As mentioned before, these protocols can provide user anonymity (as required, for example, in IPSEC) by including the user identity under the public key encryption.

3.5 Proof of Theorem 1

We start by formally describing the way a forger F attacks the Encrypted Challenge Response protocol as a probabilistic experiment, involving the user U, the server S and a forger F (this follows the definition of basic security from Section 2.4. This experiment proceeds as follows.

1. (a) A password is chosen uniformly at random from a dictionary $\mathsf{spwd} \leftarrow \mathcal{D}$.

   (b) The server's keys are generated with the key-generation algorithm of the encryption scheme (with security parameter k), $(\mathsf{sk}_S, \mathsf{pk}_S) \leftarrow \text{GEN}(1^k)$.

   (c) The user is given spwd and $\mathsf{ppwd} = MD(\mathsf{pk}_S)$. The server is given the password spwd and the key-pair $(\mathsf{sk}_S, \mathsf{pk}_S)$.
2. The forger F is given the server's public key pk_S. Also, the server is run for v times and generates first-flow messages $x_i = (\mathsf{pk}_S, \mathsf{r}_i),\ i = 1 \ldots v$.
3. The forger F is given the messages x_i, and it generates m (possibly other) alleged first-flow messages $x'_i = (\mathsf{pk}_S, \mathsf{r}'_i),\ i = 1 \ldots m$. (note that since MD is collision-intractable, we insist that all the first-flow messages generated by F consist of the right public key pk_S.)

   After generating each x'_i, the forger sends it to the user U, who responds with the reply

   $$c_i = \text{ENC}_{\mathsf{pk}_S}(f(\mathsf{spwd}\,; U, S, \mathsf{r}'_i))$$
4. The forger F is given the replies c_i, and it generates v (possibly other) replies $c'_i,\ i = 1, \ldots, v$. After generating each reply c'_i, the forger F is told whether or not it is accepted by the server.

A reply c'_i is considered *successful* if it is different than all the replies of the user ($c'_i \neq c_j$ for all $j = 1 \ldots m$), and yet it is accepted by the server (namely, $\text{DEC}_{\mathsf{sk}_S}(c'_i) = f(\mathsf{spwd}\,; U, S, \mathsf{r}_i)$). The forger F is said to (m, v)-break the protocol if it generates at least one successful reply c'_i.

To prove Theorem 1 we show that as long as the encryption scheme $\mathcal{E}$ resists one-ciphertext verification attacks with security ϵ, then no forger F can (m, v)-break the protocol with probability of more than $v/|\mathcal{D}| + \epsilon'$, where $\epsilon' = \epsilon \cdot m \cdot v$.

We prove this by way of contradiction. Namely, we show that if there exists a forger F which (m, v)-breaks the protocol with probability of more than $v/|\mathcal{D}| + \epsilon'$, then there exists an adversary A which has advantage of more than $\epsilon = \epsilon'/mv$ in guessing the correct answer using one-ciphertext verification attacks on $\mathcal{E}$. We start with the following simple lemma

Lemma 1 *If there exists a forger F which (m, v)-breaks the protocol with probability of more than $v/|\mathcal{D}| + \epsilon'$, then there exists a forger F' which $(m, 1)$-breaks the protocol with probability of more than $1/|\mathcal{D}| + \epsilon'/v$.*

Proof sketch: The forger F' executes the program of the forger F, but it picks at random an index $i \in \{1, \ldots, v\}$ and only execute the i'th impersonation attempt of F's with the server. In all the other attempts, F' itself plays the role of the server in Step 2 and declares the attempts unsuccessful in Step 4.

Then, with probability of $1/v$, i will be the index of the first successful reply of F, in which case the execution up to (and including) this reply will be consistent with the execution that would have resulted by letting F execute all its impersonation attacks. Thus, we get

$$\begin{aligned}\Pr[F'\ (m,1)\text{-breaks the protocol}] &\geq \frac{\Pr[F\ (m,v)\text{-breaks the protocol}]}{v} \\ &\geq 1/|\mathcal{D}| + \epsilon'/v\end{aligned}$$

■

Assume therefore that there exists a forger F' which $(m, 1)$-breaks the protocol with probability of $1/|\mathcal{D}| + \epsilon'/v$. Without loss of generality we can assume that F' never tries to send to the server a reply that came from the user (namely, we have $c'_i \neq c_j$ for all $j = 1 \ldots m$). We can assume this since other replies cannot add to its success probability. We will show that there exists a polynomial time adversary A which mounts a one-ciphertext verification attack against the encryption scheme and for which

$$\begin{aligned}&|\Pr[A \text{ guesses "encryption of } x" \mid c = \text{ENC}_{PK}(x)] \\ &- \Pr[A \text{ guesses "encryption of } x" \mid c = \text{ENC}_{PK}(s)]| \geq \epsilon'/vm\end{aligned}$$

Description of A. The algorithm A gets a public key pk as an initial input, and then it executes the forger algorithm F', playing the roles of the server and the user as follows: It picks a random password $\mathsf{spwd} \in \mathcal{D}$, and a random index $\ell \in \{1, \ldots, m\}$. It then runs F on input pk. First, it picks at random a challenge r_1 and gives F' the first-flow message $x_1 = (\mathsf{pk}, \mathsf{r}_1)$.

The forger algorithm F' generates the m modified queries $x'_j = (\mathsf{pk}, \mathsf{r}'_j),\ j = 1 \ldots m$. Now A sets $x = f(\mathsf{spwd}\,; U, S, \mathsf{r}'_\ell)$

and asks for an encryption of x. It gets back a string c which is either an encryption of x under pk, or an encryption of a random string s. Next, A answers the queries of F' as follows:

- For $j = 1 \ldots \ell - 1$, it answers the query $x_j = (\mathsf{pk}, \mathsf{r}'_j)$ with $c_j = \text{ENC}_{\mathsf{pk}}(f(\mathsf{spwd}\,; U, S, \mathsf{r}'_j))$.
- It answers query x_ℓ by setting $c_\ell = c$.
- It answers queries $j = \ell + 1 \ldots m$, by picking a random k-bit strings s_j and setting $c_j = \text{ENC}_{\mathsf{pk}}(s_j)$.

Then F' generates the modified second-flow response c'_1 (Recall that we assume that $c_j \neq c'_1$ for all j, and in particular $c'_1 \neq c_\ell = c$). Now A uses its ciphertext verification query to ask whether $\text{DEC}_{\mathsf{sk}}(c'_1) = f(\mathsf{spwd}\,; U, S, \mathsf{r}_1)$. If the answer is yes, A guesses that c is an encryption of x. Otherwise, it guesses that c is an encryption of a random string s.

Analysis. consider the mental experiment in which F' is run as in the attack, but when some of its first-flow messages x'_i are answered as in the real attack, while others are answered by $c_j = \text{ENC}_{\mathsf{pk}_S}(s_j)$ for random k-bit strings s_j. For every $j \in \{0, 1, \ldots, m\}$, denote by p_j the success probability of the forger F' in the case that the first ℓ queries are answered as in the real game and the last $m - \ell$ are answered by encryption of random strings. Then $p_m \geq 1/|D| + \epsilon'/v$, since this is the success probability of F' in the real attack. Moreover, we have

Proposition 1 $p_0 \leq 1/|\mathcal{D}|$

Proof sketch: Let $(\mathsf{pk}_S, \mathsf{r}_1)$ be the server's first-flow message and let c'_1 be the forger's second-flow response. Then there exists at most one password $\mathsf{spwd}_1 \in \mathcal{D}$ such that $\text{DEC}_{\mathsf{sk}_S}(c'_1) = f(\mathsf{spwd}_1\,; U, S, \mathsf{r}_1)$ (this is because $f(\cdot; y)$ is one-to-one for all y). Moreover, since c' is independent of the real password of the user (as the forger never gets to see anything that that depends on this password), then the success probability of F' is certainly bounded by the probability that the real password of the user is spwd_1, which is at most $1/|\mathcal{D}|$. ■

Finally, note that if A picks the index $\ell \in \{1, \ldots m\}$ then it will guess that c is an encryption of x with probability p_ℓ in the case that c is indeed an encryption of x and probability $p_{\ell-1}$ otherwise. Thus we get

$$\begin{aligned}
&\Pr[A \text{ guesses ``encryption of } x\text{''} \mid c = \text{ENC}_{\mathsf{pk}}(x)] \\
&- \Pr[A \text{ guesses ``encryption of } x\text{''} \mid c = \text{ENC}_{\mathsf{pk}}(s)] \\
&= \sum_{i=1}^{m} \Pr[\ell = i] \cdot (p_i - p_{i-1}) \\
&= \frac{1}{m} \sum_{i=1}^{m} (p_i - p_{i-1}) \;=\; \frac{1}{m}(p_m - p_0) \;\geq\; \epsilon'/vm
\end{aligned}$$

■

3.6 Semantic security is not sufficient

Recall that for the proof of Theorem 1 we need to assume that the encryption scheme resists some (weak) form of chosen ciphertext attack. We comment that this requirement is needed for the Encrypted Challenge Response protocol, even if the function f is assumed to have additional cryptographic properties. Specifically, it can be shown that there exists an encryption scheme which preserves secrecy (but is not resilient to chosen ciphertext attack), and for which this protocol is vulnerable to a password-guessing attack, regardless of the choice of the function f.

To see this, let ENC be a semantic secure encryption that encrypts bit-by-bit (e.g., the encryption scheme in [10]). To attack the protocol that uses this scheme, the forger F records the server's message $x = \mathsf{pk}_S, \mathsf{r}$ and intercepts the user's response $c = \text{ENC}_{pks}(f(\mathsf{spwd}\,; U, S, \mathsf{r}))$. It then modifies the response to get c' where c' is the same as c, except that the encryption of the last bit of $f(\mathsf{spwd}\,; U, S, \mathsf{r})$ is replaced by an encryption of the bit '0'. The modified response is sent by the attacker to tS. Depending on whether the server accepts or not, the forger now knows the least significant bit of $f(\mathsf{spwd}\,; U, S, \mathsf{r})$, and it can use this to eliminate (approximately) half of the passwords in the dictionary. This can be repeated with different challenges, until only a single password remains in the dictionary.

One should note that as opposed to an on-line password-guessing attack, this attack only requires about $\log |\mathcal{D}|$ attempts before the password is revealed (e.g., 20 attempts for a dictionary of one million passwords).

4 Resilience to password-guessing attacks implies key-exchange

It is interesting to note that although in a password setting there is a *shared secret* (albeit, a weak one) between the user and server, all the strong password mechanisms proposed in the literature employ public key techniques (e.g. [11, 4]). Below we explain this phenomenon by proving that public key primitives are indeed necessary for password protocols which resist password-guessing attacks.

In order to show that "public key primitives are needed for secure password protocols" we prove that given any secure password protocol one can use it (without further cryptographic functions) to implement a key-exchange protocol (where parties that do not share any initial secret can exchange a fresh secret via a public and authenticated conversation as in the case of the Diffie-Hellman protocol). This shows that protecting passwords from off-line guessing attacks is at least as hard as designing key exchange protocols and that the corresponding public-key like techniques are unavoidable. Moreover, using a result of Impagliazzo and Rudich [13], one can show that it is implausible to build these protocols out of conventional symmetric primitives such as block ciphers, hash functions or pseudorandom functions, as such a construction would imply a major breakthrough in complexity theory. Formally, we prove the following theorem:

Theorem 2 *Any password authentication protocol which is secure against off-line password-guessing attacks, can be used in order to implement a secure key-exchange protocol.*

Proof sketch. Assume that we have any password protocol which is secure against off-line password-guessing attacks. In particular, this means that if the parties use a dictionary $\mathcal{D}$ then a passive eavesdropping adversary (which only listen on the lines) cannot guess the user's password with probability significantly larger than $1/|\mathcal{D}|$. In particular, if the parties are using a dictionary of size 2, then the adversary cannot guess the user's password with probability significantly larger than a half.

Recall that to describe a secure key-exchange protocol, it is sufficient to show how two parties can exchange one bit

over a public channel, in such a way that a passive eavesdropper cannot guess this bit with probability significantly larger than a half. If we have a secure password protocol as above, this can be done as follows:

In order to exchange a secret bit, the two parties use the password protocol with a dictionary of size 2 (we assume without loss of generality that the dictionary is $\mathcal{D} = \{0, 1\}$). To exchange a bit, each of the parties chooses at random a password from the dictionary, and then they execute the password protocol with one party playing the server and the other playing the user. It follows from the security of the password protocol that this execution succeeds if and only if they both choose the same password.

If the execution succeeds, then their secret bit is the password that they chose. Otherwise, they choose new passwords and try again. Since the protocol resist password-guessing attacks, then an eavesdropping adversary cannot guess the password that was used in a successful execution, and so the exchanged bit is indeed secret. After expected 2 trials, the parties will be able to exchange the secret bit. ■

5 Public Passwords

In order to allow for the use of our secure protocols in the cases where the client's machine cannot verify the authenticity of the server's public key, we suggest to provide the user with a hashed version of this certificate. We call this information a "public password". This results in an extension to the usual human-password paradigm, where the user carries not only a secret password, but also a public password. The latter *requires no secrecy* protection but requires integrity. The public password should be short enough so that a human user is able to recognize it if displayed, or even to type it in if requested to do so, but it does not need to be memorized and can be safely written down on a piece of paper, a sticker, a plastic card, etc.

In our applications the public password serves as "hand-held certificate" for a public key, which the user can conveniently carry with him. Whenever presented with the actual public key (e.g., after being transmitted to the user's terminal) the user can verify the validity of the public key against the hand-held certificate. This enables a human user to participate in protocols that otherwise would be impossible to carry out without a memory device. This notion may be useful in other scenarios as well before a reliable global public key infrastructure becomes available. (This solution is suited, for example, to credit-card applications, where the hand-held certificate can be recorded on the credit-card itself.) Moreover, even after such public key infrastructure will be in place, hand-held certificates may be useful as a supplement to the trust level offered by other mechanisms such as certification authorities (e.g. X.509), distributed directories (e.g.,Secure DNS), and others.[7]

Below we elaborate on some implementation issues of public passwords. As said, we use public passwords as digests of public keys. For this to be of any use, there should be no feasible way to find a second public key that hashes to the same public password. Thus, the length of the public password depends on the amount of trust that we have on the party generating these public keys. If this party (user or server) is trusted not to look for collisions in the hash function during the process of key generation, then the public password needs only to resist "second preimage attacks". That is, it should be infeasible – given a public key pk – to find another pk' such that $H(\mathsf{pk}) = H(\mathsf{pk}')$. In this case, a public password of 60 to 80 bits will suffice. If the generator of the public key is not trusted then H needs to be fully collision resistant and then its output should be in the 120-160 bit range. For the uses in this paper it seems reasonable to assume that the key-generation process is done properly.

5.1 Representation and identification of public passwords

Even though public passwords are short enough to be carried by a human being, they usually represent unstructured strings. Thus, for a user to be able to read, recognize, and type the public password, it is advisable to have a user-readable format for these passwords. A representation for mapping arbitrary binary strings into easy-to-read (and write) words was introduced in the context of one-time passwords [12]. This solution defines a dictionary of 2048 words (mostly English words, 2 to 4 letter long) and a mapping of each 11-bit string to a different word in the dictionary. Thus a 66 bit string is represented by 6 words from that dictionary, e.g. `moss mont sit rear rage pit`.
Such a representation could be used by a public password system as well. Of course, many other representations are possible, e.g using just alphanumerics (without case distinction) would require about 12 characters to represent 60-bit strings, e.g.: `a6et qw29 hzjv`.

Another difference between the public passwords and the secret ones is that in our applications there is no need for the user to type in the public password. Consider again the case of a public password ppwd consisting of a hash of a public key pk. The latter is stored in some remote machine and sent over the network to the local terminal where the user is working. The terminal computes the hash of the public key and then compares the computed value with the public password ppwd. This can be done by having the user enter ppwd, but also by just displaying the computed hash on the screen and asking the user to approve it. Moreover, many users may be able, after some time, to recognize the right value without even carrying it with them.

In this case, however, it is important that a user carefully checks for the validity of the displayed value. Thus, the user-interface should be designed carefully to avoid the tendency of users to answer every question by simply hitting the `Enter`-key. An example of one such possible user-interface is to display five strings to the user, one of which is the correct password, and have the user type the number corresponding to the right public password. For instance, if the public password is `moss mont sit rear rage pit`, the user may be presented with:

```
1. eddy weak half net ohio ok
2. moss mont sit rear rage pit
3. ivan laud loy an gal but
4. bloc ave fire grad beef aye
5. vary hone ton limb pry stew
```

to which he is required to answer with '2'. We stress that it is important that the user does not blindly decide by a matching prefix or so, as this would help an attacker in delivering a public key of its choice. Thus, in the above example it may

[7] One can envision applications of this notion where users carry hand-held certificates for public keys of a few entities with which they interact frequently (e.g., the public key of their administrative domain, etc.), thus avoiding the need to rely on public-key infrastructure in every connection with these entities. In a sense, this is somewhat similar to carrying a short list of often used telephone numbers to avoid the need to refer to the directory for every phone call.

be more effective to present the user with strings that are actually visually related to each other (and only one being the right value). There may be other graphical encodings of bits that will be even more easily recognizable by the user. Some recent work on a similar recognition problem can be found in [8].

References

[1] M. Bellare, R. Canetti and H. Krawczyk, "A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols", *Proceedings of the Thirtieth ACM Symposium on the Theory of Computation (STOC)*, 1998, pp. 419–428.

[2] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, "Relations Among Notions of Security for Public-Key Encryption Schemes", *Advances in Cryptology - CRYPTO'98 Proceedings*, Lecture Notes in Computer Science Vol. 1462, H. Krawczyk, ed., Springer-Verlag, 1998, pp. 26–45.

[3] M. Bellare, and P. Rogaway, "Optimal Asymmetric Encryption – How to encrypt with RSA", *Advances in Cryptology - EUROCRYPT'94 Proceedings*, Lecture Notes in Computer Science Vol. 950, A. De Santis ed, Springer-Verlag, 1995.

[4] S. M. Bellovin and M. Merritt, "Encrypted Key Exchange: Password- Based Protocols Secure Against Dictionary Attacks", *Proceedings of the IEEE. Symposium on Research in Security and Privacy*, Oakland, May 1992.

[5] S. M. Bellovin and M. Merritt, "Augmented Encrypted Key Exchange: a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise", *Proceedings of the First ACM Conference on Computer and Communications Security*, 1993, pp. 244-250.

[6] R. Cramer and V. Shoup, "A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack", *Advances in Cryptology - CRYPTO'98 Proceedings*, Lecture Notes in Computer Science Vol. 1462, H. Krawczyk, ed., Springer-Verlag, 1998, pp. 13–25.

[7] D. Dolev, C. Dwork, and M. Naor. "Non-malleable cryptography". *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 542-552, 1991.

[8] I. Goldberg, H. Finney and R. Levien. "Visual Fingerprints" and "Snowflakes". `http://www.cs.berkeley.edu/~iang/visprint.html`

[9] O. Goldreich, S. Goldwasser and S. Micali. "How to Construct Random Functions", *Journal of the ACM*, Vol. 33, no. 4, 1986, pp. 792-807

[10] S. Goldwasser, and S. Micali. "Probabilistic Encryption", *Journal of Computer and System Sciences*, Vol. 28, 1984, pp. 270-299.

[11] L. Gong, M. Lomas, R. Needham, and J. Saltzer, "Protecting Poorly Chosen Secrets from Guessing Attacks", *I.E.E.E. Journal on Selected Areas in Communications*, Vol. 11, No. 5, June 1993, pp. 648-656.

[12] N. Haller, "The S/KEY One-Time Password System", RFC 1760, Feb. 1995.

[13] R. Impagliazzo and S. Rudich. "Limits on the provable consequences of one-way permutations". In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, 1989, pages 44-61.

[14] D. Jablon, "Strong Password-Only Authenticated Key Exchange". *Computer Communication Review*, ACM SIGCOMM, vol. 26, no. 5, pp. 5-26, October 1996.

[15] C. Kaufman, R. Perlman, and M. Speciner, "Network Security," Prentice Hall, 1997.

[16] H. Krawczyk, "SKEME: A Versatile Secure Key Exchange Mechanism for Internet,", *Proceedings of the 1996 Internet Society Symposium on Network and Distributed System Security,* Feb. 1996, pp. 114-127.

[17] L. Lamport, "Password authentication with insecure communication," *Comm. of the ACM,* Vol. 24 Number 11, Nov 1981, pp. 770-772.

[18] S. Lucks, "Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys", *The Security Protocol Workshop '97*, Ecole Normale Superieure, April 7-9, 1997.

[19] A. Menezes, P. Van Oorschot and S. Vanstone, "Handbook of Applied Cryptography," CRC Press, 1997.

[20] S. Patel, "Number Theoretic Attacks On Secure Password Schemes" *IEEE Symposium on Security and Privacy*, Oakland, California, May 5-7, 1997.

[21] C. Rackoff and D. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack", *Advances in Cryptology - CRYPTO'91 Proceedings*, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed, Springer-Verlag, 1991.

[22] M. Steiner, G. Tsudik, and M. Waidner, "Refinement and Extension of Encrypted Key Exchange", *Operating Systems Review*, vol. 29, Iss. 3, pp. 22-30 (July 1995).

[23] T. Wu, The Secure Remote Password Protocol, in *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, San Diego, CA, Mar 1998, pp. 97-111.