Can "Something You Know" Be Saved?

Baris Coskun[1] and Cormac Herley[2]

[1] Polytechnic University, Brooklyn, NY
[2] Microsoft Research, Redmond, WA

Abstract. "Something you know," in the form of passwords, has been the cornerstone of authentication for some time; however the inability to survive replay attack threatens this state of affairs. While "something you know" may always be used in addition to "something you have" we examine whether it can be salvaged as the solo factor for authentication. A recent surge of interest in Challenge Response authentication schemes raises the question whether a secret shared between the user and the server can allow secure access even in the presence of spyware.

Our conclusion is negative. Assuming only a limit on the amount that a user can remember and calculate we find that any scheme likely to be usable is too easily brute forced if the attacker observes several logins. This is true irrespective of the details of the scheme. The vital parameter is the number of bits of the secret involved in each bit of the response. When this number is too low the scheme is easily brute-forced, but making it high makes the scheme unworkable for the user. Our conclusion is that single factor "something you know" schemes have a fundamental weakness unless the number of logins the attacker observes can be restricted.

Keywords: Authentication, passwords, challenge response.

1 Introduction

Authentication is commonly implemented by requiring a user to provide proof of one or more of:

- Something you know (*e.g.* a password)
- Something you have (*e.g.* a smartcard)
- Something you are (*e.g.* a fingerprint).

Passwords have enjoyed a long run as the dominant means of authentication. An active web user today will access financial institutions, social networking, and email accounts using passwords. It is not uncommon for users to have twenty or more password-protected accounts, and to type passwords several times per day. With this success has come a host or problems and attacks. Users notoriously choose weak passwords, potentially opening the door to brute-force attacks. The Phishing phenomenon has shown that users can be lured into divulging their passwords to sites masquerading as the real login server. Malicious spyware such

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 421–440, 2008.

as a keylogger can capture the password and thereby afterward allow an attacker access to the protected account.

Despite these problems passwords are still almost universally used for account access, even when assets of considerable value are protected. A majority of the large banks and financial institutions in the US appear to use password only access. The reasons for the success are clear: passwords are a simple and well-understood technique. They allow institutions to offer users 24/7 access to their accounts from any browser. No special hardware or training is required, and they rely on memory only. In addition it is argued in [9] that losses from password stealing attacks may not be as high as often assumed.

Our focus in this paper is on the failure of passwords to resist replay attack. That is, a single login from a spyware infected machine gives an attacker everything he needs to gain access to an account. If an unsuspecting user accesses their bank account from an infected internet cafe machine it is easy for the spyware to gather the userID and password and then access the account when the attacker chooses.

There has been a great deal of password related work recently, which we attempt to review in Section 2. Our goal in this paper is to evaluate mechanisms for access control from untrusted machines that rely only on the memory and calculating power of the user. Examples are the recently proposed scheme of Weinshall [17], the Virtual Password work of Lei *et al.* [12], the picture authentication work of Pering *et al.* [15], and the work of Cheswick [3]. Each of these approaches attempts to allow users to login securely from untrusted machines. That is, spyware running on the machine and observing a login should not be able to use that information to gain access to the account. To be useful, the scheme should allow several logins from the same machine without giving the attacker enough information to gain access. Golle and Wagner [7] for example recently showed that the Weinshall scheme could be broken after observing 7 or so logins. We show how to break the Virtual Passwords scheme of [12] in Section A.

The constraints that we impose are that the scheme should rely only on the user's memory and calculating ability. As we will see in Section 2 none of the solutions proposed so far to the untrusted login problem possess all of the desired features mentioned above. Our goal is to determine whether it is possible to construct a scheme which satisfies all of the requirements. For example, can the break in Weinshall's scheme be fixed or is it a fundamental hole? The pattern of progress on this question has been of suggested solutions, such as [17], followed by rebuttals, such as [7]. This is an unsatisfying state of affairs. The problem is important enough that researchers often return to it in hopes of a solution. Yet, in breaking such a scheme we seldom find out whether that particular scheme was flawed or whether there is a fundamental limitation that prevents us from designing a "Something You Know" scheme that will withstand determined attack. This paper demonstrates that the problem is truly a fundamental one: Weinshall [17] and the other schemes commit the common error of failing to involve enough bits of the secret in calculating each bit of the output. For any scheme that fails to do so a brute-force attack can determine the secret given

enough observed logins. The only ways to avoid this is to involve more bits of the secret per output bit, increase the size of the secret, or restrict the number of logins observed. But we show that increasing either the secret size or number of bits used places an insupportable burden on the user: for example to involve each bit of a 60-bit secret in each bit of a 20-bit login would require the user to make two binary decisions per second at a sustained rate for 10 minutes in order to login.

2 Related Work

2.1 Challenge Response Authentication

A scheme introduced by Weinshall [17] assigns a user thirty images to memorize. When logging in the user is presented with a palette containing 8×10 images. Starting at the top left corner he calculates a path by moving down or to the right depending on whether the current image is one of his thirty images. On reaching the bottom or right edge of the palette the path exits and he enters a 2-bit number to indicate the exit point. This is performed 11 times for a 22-bit login. The scheme was broken using SatSolver by Golle and Wagner [7].

Pering *et al.* [15] suggest having the user upload a number of his own images. When logging in he is presented with a palette of four images, one of which is his. He selects his image (thus effectively giving away 2 bits) and repeats 10 times for a 20 bit login. Clearly the user must upload to the server at least $10\times$ as many images as the attacker will observe logins.

Lei *et al.* [12] propose a scheme to secure users' passwords in spyware infected environments. The user must calculate response symbols that are derived from symbols of a fixed password and a challenge symbol string using modular arithmetic. The complexity of the calculation appears considerable. The authors suggest a helper application can be used to assist the user in performing the calculations. We show how this scheme can be broken in Section A. Cheswick [3] explores the general feasibility of allowing multiple logins from a single shared secret. He proposes a number of approaches, but draws no definite conclusion on whether the goal is attainable or not.

2.2 Logging in from Untrusted Machines

Pashalidis and Mitchell [14] propose a single sign-on system as a means of evading spyware on an untrusted machine. Credentials are stored in the cloud and a challenge response authentication is used to grant access. The user is assigned a secret string, and when accessing the credentials is challenged to produce the characters at three randomly chosen positions in the string.

Florêncio and Herley [5] propose a proxy-based solution where the user maps the password in a fashion that is unmapped by a MITM proxy before forwarding to the login server. The same authors suggest [2] a simple trick that allows a user to evade current generation keyloggers when entering a password on an untrusted machine.

Lim [13] proposes a scheme to prevent spyware from capturing screenshots. The user enters data from an on-screen keyboard. Rather than comprising a single image the on-screen keyboard is formed from several images displayed in rapid succession. A single screen shot at the time of the mouse click will not tell the spyware what key was selected, while capturing many images will force the spyware to consume resources thereby risking discovery.

One-time password systems such as S/Key [11,8] resist replay. SecurID [1] is a well known commercial product that generates time-evolving one-time codes on a keychain device that match codes generated at the server. In an enhancement of [5] Florêncio and Herley [4] propose a proxy-based solution where the user enters an encrypted version of the true password that is decrypted by the proxy before forwarding to the login server. As with any one-time password system the user must carry either a list of the one-time passwords, or a device that calculates them. The scheme of [4] differs from [11,8,1] in that it works with existing password servers without modification.

2.3 Alternatives to Passwords

Passwords are so widely used, and attacks on them so varied that a large literature has grown around addressing these attacks. We can give only a small sample of recent password related work. Graphical passwords [10,16] address the oft-cited problem with passwords that users have too many passwords to remember and often make weak easily guessed choices. This approach promises to improve the password strength and memorability, it of course does nothing to address the replay attack. Florencio *et al.* [6] argue that passwords strength is of limited importance when password stealing techniques such as phishing and keylogging are the main threats.

Two-factor authentication is the practice of requiring possession of a piece of hardware, or a biometric before allowing login. While far more secure tokens often require an issuing authority, and require that the user carry something. Our work can be seen as an examination of whether one factor schemes can ever truly resist replay.

3 Challenge-Response Schemes

Shared secret (*i.e.* "Something you know") schemes require that a user prove that she knows the secret before access is granted to an account. Passwords are the simplest case, since entry of the password, $\mathbf{P}$, causes the server to conclude that the request for access has come from the correct user. However, since the secret is not dynamic, a single observation suffices to allow an attacker to break the system. One-time password systems, by contrast, deny an attacker useful information, even assuming he observes a login. Here, the user possesses a list, rather than a single password, and enters the i-th password, $\mathbf{P}_i$, on demand from the server. The successive passwords $\mathbf{P}_i$, can be derived from a single secret, as with S/Key [11,8], dynamically generated on-the-fly as with SecureID

[1], encrypted versions of a single static password as in [4] or just a pre-computed set of random strings. In each case, of course, the user must carry the list of one-time passwords (or the device that calculates them). We consider it unreasonable to expect that a user will commit to memory a series of passwords each of which will be used only once.

With a challenge response scheme the user again shares a secret $\mathbf{S}$ with the server, but to login demonstrates that she knows the secret without revealing the secret itself. Instead of delivering the entire secret, the user delivers something derived from the secret in response to a challenge from the server. Thus the server produces a random challenge $\mathbf{C}_i$ and requests that the user return $f(\mathbf{S}, \mathbf{C}_i)$, where $f()$ represents a calculation that the user must perform (of which more below). The server will produce a new challenge for each login, and hence the attacker cannot simply replay an observed response, since

$$f(\mathbf{S}, \mathbf{C}_i) \neq f(\mathbf{S}, \mathbf{C}_j).$$

The basic outline of the three authentication schemes discussed is given in Figure 6.

While replay may not be possible on a challenge response scheme there is still the possibility that given several observations $f(\mathbf{S}, \mathbf{C}_i)$, for $i = 0, 1, 2 \cdots$ the attacker may be able to determine the secret $\mathbf{S}$. Clearly this depends on the function $f()$. Ideally, an attacker should be unable to determine the secret even after observing many logins. If such a scheme exists it could be enormously beneficial. It would share with passwords the fact that it is memory based and requires no hardware. And yet it would entirely solve phishing, keylogging and other password stealing attacks.

3.1 General Setup and Notation

We will assume that during a registration process the user and server agree to share an N-bit secret $\mathbf{S}$. For the i-th login the user provides the userID and receives a randomly chosen challenge $\mathbf{C}_i$ from the server. Given this challenge, she must calculate, and deliver to the server, the M-bit response

$$\mathbf{R} = f(\mathbf{S}, \mathbf{C}_i).$$

This model is sufficiently general to cover the major existing challenge response proposals *e.g.* [17,3,15,12].

The Secret. The size, N, of the secret space must be large. Recall that $\mathbf{C}_i$ is known to the attacker, and of course $f()$ must be public. If N is small enough the attacker might just list the response

$$\mathbf{R}' = f(\mathbf{S}', \mathbf{C}_i)$$

for each of the 2^N possible secrets. Any $\mathbf{S}'$ for which $\mathbf{R}' = \mathbf{R}$ is a candidate for the true secret. There would be only 2^{N-M} candidate secrets after observing a

single response. Subsequent logins would narrow the field further. The key to avoiding this attack is that 2^N must be too large for an attacker to list. The tradeoff, of course, is that N must be small enough for the user to remember and perform calculations on. A 256-bit secret space will resist enumeration, but this is equivalent to a 77-digit PIN, and is probably far too much for a user to remember. How many bits a user can be expected to remember and perform calculations on is somewhat representation dependent. In theory users might be able to remember an $N = 80$ bit secret (*i.e.* equivalent to an 24-digit PIN), but it is hard to argue that they could also reliably perform calculations on such a large secret. Weinshall [17] makes both the memorization and calculation tasks simpler by making the user memorize the secret in a set membership format. At the lower end if a machine can evaluate the response to 2^{10} challenges per second (this is approximately the rate we achieved with an implementation of the Weinshall scheme [17]) then even a secret space of size $N = 37$ can be exhaustively searched by a single machine in a single year. Since enumerating responses to a challenge is a task easily divided among many machines we probably have to consider secrets on the order of 50 bits as the minimum that can even be considered for successive logins on a compromised machine. Thus we get a probable range for the secret space, bounded below by 50 bits as the minimum to resist enumeration and above by 80 bits as probably the most that a human can be expected to remember and calculate on.

The Response. Of course $\mathbf{R}$ should be drawn from a large enough space to make random guessing unlikely to succeed. For example, if $M = 20$ then a single random guess at $\mathbf{R}$ has only a one in a million chance of succeeding. Since the response must be entered using keyboard and/or mouse $\mathbf{R}$ is generally delivered as a series of T symbols $\mathbf{R} = R(0)R(1)R(2)\cdots R(T-1)$, where each $R(t)$ is a k-bit symbol and $kT \approx M$. For example, the response to the challenge might be a 6-digit number since $\log_2(10) \cdot 6 \approx 20$. In [17] the response is a series of $T = 11$ 2-bit symbols, giving a 22-bit effective login.

The Challenge. The challenge is the random component that makes each response different. The form that it takes depends on the form in which the secret is stored. The main requirement is that the server have a large enough suite of challenges. If there are fewer than 2^M challenges then we have unnecessarily reduced the size of the output response space (making the attackers guessing task easier).

The Calculation. Without loss of generality we'll say that each response symbol is produced by a separate calculation function: $R(t) = f_t(\mathbf{S}, \mathbf{C}_i)$. Thus overall

$$\mathbf{R} = [f_0(\mathbf{S}, \mathbf{C}_i) f_1(\mathbf{S}, \mathbf{C}_i) f_2(\mathbf{S}, \mathbf{C}_i) \cdots f_{T-1}(\mathbf{S}, \mathbf{C}_i)].$$

Each of the $f_t()$ represents a calculation the user must perform, using the secret and the challenge, to produce the response symbol $R(t)$. In general it will involve fewer than N bits of the secret. This is so, since if each bit of $\mathbf{R}$ depends uniformly at random on each bit of $\mathbf{S}$ the user must carry out at least $M \cdot (N-1)$ binary

decisions (see Section 4.4). For an 80-bit secret and a 20-bit response, this would be 1600 binary decisions for a single login. Even if a user could reliably make 2 decisions per second the login would take 13.3 minutes, which is absurd. Thus, we will assume, in general, that $U \leq N$ of the bits of $\mathbf{S}$ are used in the calculation $f_t()$ of each of the response symbols $R(t)$. The lower U the easier the task for the user. It is clear that $U = N$ represents an insupportable burden. However, as we will see in Section 4, making U too small invites a divide-and-conquer attack. U will play a vital role in the tradeoff between usability and security.

We emphasize that the calculation $f(\mathbf{R}, \mathbf{C}_i)$ must be simple enough for a user to perform quickly and accurately. The user must not employ any calculating devices hosted on the untrusted machine. For example, using even a software calculator on the untrusted machine (such as the scheme of [12] suggests) would be unacceptable, since the attacker would have access to the input as well as the output.

We also rule out using the assistance of a calculator on a cell phone, mp3 player or other device, but for a different reason. If the user has access to a cell phone it makes more sense to carry a list of one-time passwords than to perform a challenge response calculation. Our goal is to determine whether a memory alone challenge response scheme can resist attack.

Example Schemes. The described notation and framework are applicable to all challenge-response authentication schemes although the mapping function is different in each scheme. For instance in Weinshall's scheme [17] briefly described in Section 2, the shared secret is the set of 30 images selected among 80. Here the size of the secret is $N = \log_2 \binom{80}{30} \approx 73$ bits. C_i is the permutation of the 80 images which is currently being displayed on the panel and the mapping function M is the mental path that the user is supposed to follow. The response $R(t)$ is the label through which the path drawn by the user exits the panel. Since $R(t)$ can be either 0,1,2 or 3, it is a $k = 2$ bit response. Finally, the server requires 11 challenge-response rounds, which makes $T = 11$.

3.2 Two Trivial Solutions

Hand Over the Bits Unmodified. A trivial solution is to have $f_t()$ just select k bits of the secret. For example, the first time the user logs in she might be asked to enter the first M bits of the secret $\mathbf{S}$, the second time the next M and so on. This has the merit of being simple, but this is good for only N/M logins on the same machine. After that the attacker knows the entire secret and those bits cannot be re-used. In fact this is equivalent to a one-time password system. A variant of this is used in [15], since the user gives away bits each time he indicates an image.

Challenge for Random Bits of Secret. A related alternative is to prompt the user for specific randomly chosen portions of the secret, which are delivered unmodified (this is suggested *e.g.* in [14]). For example, if the secret were remembered in the form of a 24-digit number the server might challenge for 6

randomly chosen digits of the secret (thereby giving a 20-bit login). This has the merit of again being simple, but extending the number of logins that can be achieved. However the probability that the attacker who has observed n logins possesses any particular digit is

$$Pr\{\text{Attacker knows digit}|W \text{ logins}\} = 1 - (1 - M/N)^W.$$

For example, after 8 logins he knows each digit with 90% probability. At this point he has a 53% probability of being able to answer any given challenge successfully. If he gets three attempts before the account is locked he has a greater than 90% chance of being able to successfully respond to one of the challenges.

What's Wrong with These Solutions. It is apparent that in order to withstand more than $\frac{N}{M}$ challenge-response rounds, the server should never ask for the secret bits themselves. For this purpose a carefully designed $f()$ function, which proves to the server that the user knows the secret without revealing the secret bits, is required. Such an $f()$ function should be one-way in the first place so that it should be very hard to determine the input given the output of the function. Otherwise, the scheme would be equivalent to the case where the user submits secret bits as the response.

3.3 Attack Model

First we introduce our attack model. We assume that the attacker has installed spyware on the untrusted machine and observes everything that happens there. All keystrokes, all mouse-movements, everything that goes on the screen, and all network traffic is available to the attacker.

Thus, following the general pattern of a challenge response scheme in Figure 6 the attacker will observe both the challenge from the server $\mathbf{C}_i$ and the client's response $\mathbf{R} = f(\mathbf{S}, \mathbf{C}_i)$. We further assume that the calculating function $f()$ is public, so that the secret $\mathbf{S}$ is the only thing concealed from the attacker. Since we desire that the user be able to login multiple times we will assume that the attacker observes a series of W logins and gets the MW-bit stream:

$$\mathbf{\Gamma} = \mathbf{R}_0\mathbf{R}_1\mathbf{R}_2\cdots\mathbf{R}_{W-1}.$$

Since he has seen the response the attacker can try as many off-line guesses at the secret as computation will allow. That is he can calculate

$$\mathbf{\Gamma}' = \mathbf{R}_0'\mathbf{R}_1'\mathbf{R}_2'\cdots\mathbf{R}_{W-1}',$$

for as many trial secrets $\mathbf{S}'$ as he wishes. If he finds an $\mathbf{S}'$ for which $\mathbf{\Gamma}' = \mathbf{\Gamma}$ he has not necessarily found $\mathbf{S}$. For example, after a single login (*i.e.* $W = 1$) there would be 2^{N-M} secrets that produce the same response as $\mathbf{S}$. However as the number of observed logins increases collisions decrease rapidly. When $MW > N$, *i.e.* the total number of observed bits is greater than the secret size, collisions are negligible and we do not consider them further.

We Cannot Conceal How Many Bits are Used. Recall, from Section 3.1 that the number of bits of the secret U involved in each output bit is an important parameter. This cannot be concealed from the attacker. We can measure the effective number of bits used by a given scheme as follows. Choose two secrets $\mathbf{S}$ and $\mathbf{S}^{'}$ which differ by one bit. Generate A randomly chosen challenges and count the number, B, for which the two secrets produce different responses. Thus the measured fraction of responses where the two secrets produce the same response is $(A - B)/A$. We have already seen (in the case of k-bit symbols) that the expected value is $1 - U/N \cdot (1 - 1/2^k)$. Thus if we compare the expected and actual values we can estimate U as

$$U = \frac{NB}{A} \cdot \frac{1}{1 - 1/2^k}. \tag{1}$$

Of course A and B should be large enough to generate a stable estimate of the actual fraction of responses that remain unchanged. We can use this to estimate the effective number of bits used, for example in the Weinshall [17] scheme, where we find $U_{eff} \approx 7.8$. We will see that this is the fatal weakness, not just of this scheme, but of any scheme which does not make absurd calculating requirements of the user.

4 A Brute Force Attack

We now show how secret can be identified within a reasonable time when the number of bits, U, used to calculate an output symbol is small. The weakness is that when $U \ll N$ similar secrets produce similar responses. The outline of the attack is as follows:

- When secrets are close the responses are close
- It's easy to find a secret that's close
- Once close, it's easy to get closer.

Taken together this gives that the only escape from brute-force attack is to make U large. But we show in Section 4.4 that this forces the burden of calculation on the user to be infeasible.

4.1 When Secrets Are Close the Responses Are Close

First suppose that $\mathbf{S}$ is the user's secret and $\mathbf{S}^{'}$ is an unrelated secret chosen at random. We expect that the probability that two symbols from their respective responses to $\mathbf{C}_i$ are the same is $Pr\{R(t) = R(t)^{'}\} = 1/2^k$. This merely says that all response symbols are equally likely and, for unrelated secrets, the response symbols coincide at random. However, when $\mathbf{S}$ and $\mathbf{S}'$ are close and $U \ll N$ this is no longer true: suppose they differ by e bits *i.e.* $|\mathbf{S} - \mathbf{S}^{'}| = e$. Now, since only a U-bit portion of the N-bit secret is used to calculate each response symbol this portion might include none of the different bits. For example, if $\mathbf{S}$ and $\mathbf{S}^{'}$

differed in only a single position (*i.e.* $e = 1$) we would have $R(t) = R^{'}(t)$ unless that single bit was one of the U bits used in this calculation.

In general the probability that none of the e bits where $\mathbf{S}$ and $\mathbf{S}'$ differ are included in any of the U bits used to calculate $R(t)$ is:

$$\frac{N-e}{N}\frac{N-1-e}{N-1}\cdots\frac{N-(U-1)-e}{N-(U-1)} = \left(\prod_{j=0}^{U-1}\frac{N-j-e}{N-j}\right).$$

When this occurs $R(t) = R^{'}(t)$. Otherwise the responses might still be the same with probability $\frac{1}{2^k}$. Hence the probability of having $R(t) = R^{'}(t)$ given that $|\mathbf{S} - \mathbf{S}'| = e$, can be written as:

$$p_e = \left(\prod_{j=0}^{U-1}\frac{N-j-e}{N-j}\right) + \left(1 - \left(\prod_{j=0}^{U-1}\frac{N-j-e}{N-j}\right)\right)/2^k\,. \tag{2}$$

We graph this in Figure 1. As e decreases (*i.e.* $\mathbf{S}$ and $\mathbf{S}^{'}$ get closer) p_e deviates from $\frac{1}{2^k}$ and gets closer to 1 as shown. When e is very small the probability that $R(t) = R^{'}(t)$ is almost 1. This is important since it says that the probability that the response symbols are equal increases as the distance between the secrets decreases.

Now let's examine the consequences for the attacker who gathers TW response symbols from the W observed login events. Define $\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{'})$ as the number of symbol positions in which the two response streams agree; this ranges between 0 and TW. Now, $Pr\{\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{'}) = d\}$ is binomially distributed with probability p_e given in (2):

$$Pr\{\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{'}) = d \mid |\mathbf{S} - \mathbf{S}^{'}| = e\} = B_{pdf}(d, TW, p_e).$$

In Figure 2 (a) we show how this is influenced by distance (for $N = 80, U = 10, k = 2$ and $W = 10$ logins) by showing the distributions for $e = 12$ and $e = 40$. Observe that the mean of $\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{'})$ for secrets that are distance 40 from $\mathbf{S}$ is $TWp_{40} = 100 \times 0.25 = 25$ while for secrets at distance 12 it is $TWp_{12} = 100 \times 0.413 = 41.3$. Also observe that for a given scheme (*i.e.* N, U and k fixed) the separation increases with the number of logins observed. Figure 2 (b) we show the same scheme as in (a), but now with $W = 20$ logins. Clearly it gets easier to tell secrets that are close to $\mathbf{S}$ from those that are far when the attacker observes more logins. Thus the attacker is always assisted by more observations: the more logins he observes the greater the separation between the binomials and the more reliably he can distinguish secrets that are close from secrets that are far from the true secret.

We emphasize that when $\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{'})$ is large it does not imply that $|\mathbf{S} - \mathbf{S}^{'}|$ is small, but it does mean that the probability is higher. Thus, among a large enough collection of secrets it is possible to distinguish *in probability* those that are close to the true secret from those that are far.

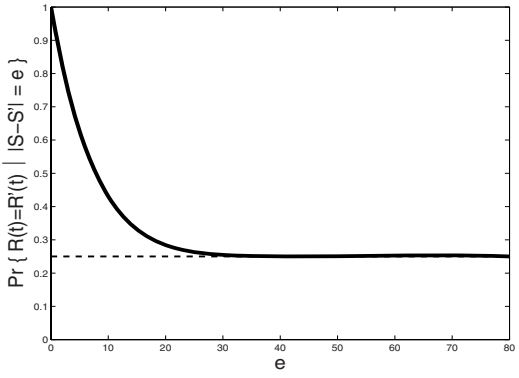

Fig. 1. Probability that an output symbol $R(t)$ is the same for two N-bit secrets that differ in e positions: $Pr\{R(t) = R(t)^{'} \mid |\mathbf{S} - \mathbf{S}^{'}| = e\}$ vs. e. We are using $N = 80$, $U = 10$ and $k = 2$. Observe that when secrets are close (*i.e.* e small) the probability of the outputs being equal is high.

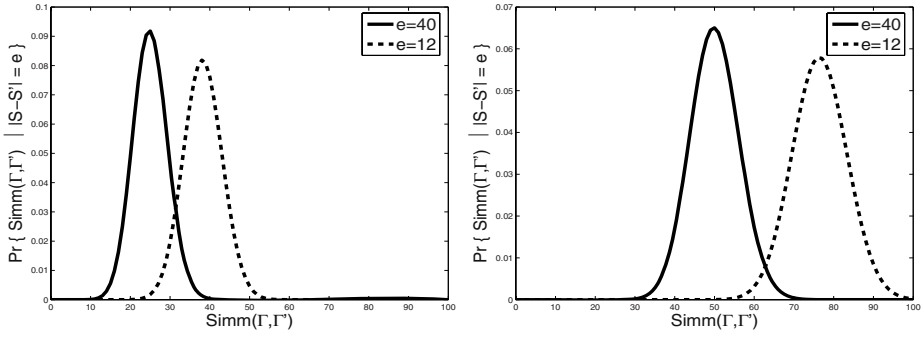

Fig. 2. Secrets that are close produce responses that are close. The expected number of positions where the responses are the same for depends on distance from the true secret. Using $N = 80$, $U = 10$ and $k = 2$ and secrets that are 40 and 12 bits from the true secret. (a) TW = 100 (*i.e.* 10 logins each with 10 2-bit symbols). (a) TW = 200 (*i.e.* 20 logins each with 10 2-bit symbols).

4.2 It's Easy to Find a Secret That's Close

If the attacker has a large collection of candidate secrets he now has a good strategy to tell secrets that are close from secrets that are far. By selecting only those points for which

$$\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{'}) \geq \text{Threshold} \tag{3}$$

we can remove from consideration those that are likely to be far away. For example, in Figure 2, if we threshold at TWp_{12}, *i.e.* the mean of the p_{12} binomial, we include a fraction $1 - B_{cdf}(TWp_{12}, TW, p_{12}) = 0.5$ of secrets that are distance 12 from $\mathbf{S}$ and only $1 - B_{cdf}(TWP_{12}, TW, p_{40}) = 0.0065$ of those that are distance

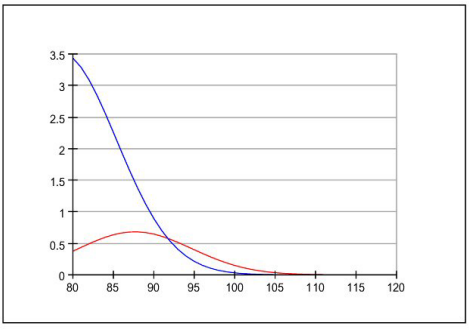

Fig. 3. Figuring out which of your neighbors take you closer to/away from the true secret. The graph shows that even the binomial pdfs with probabilities p_{e+1} and p_{e-1} are separated when e is small enough. The scheme shown is for $N = 73, U = 8.8, k = 2$ with 10 logins.

40. Thus, at a threshold of TWp_{12} the vast majority of distance-40 points are discarded, while only 0.50 of distance-12 points are. Of course there are many more secrets at distance 40 than at distance 12, *i.e.* $\binom{N}{40} \gg \binom{N}{12}$. But if the attacker starts with a large enough collection of candidate secrets $\mathbf{S}'$ he can quickly trim the collection to include only those that have a higher probability of being close. Of the 2^N total secrets in the space the number of points that satisfy (3) (*i.e.* have response streams that are close to that produced by the true secret) is given by summing the tails of the binomials multiplied by the number of points:

$$\sum_{k=0}^{N} \binom{N}{k} \cdot (1 - B_{cdf}(\text{Threshold}, TW, p_k)).$$

A total of $\binom{N}{e}$ secrets will live within an e-ball of the user's secret. Thus we should expect that if the attacker selected

$$2^N \Big/ \binom{N}{e}$$

secrets at random, at least one would be a distance e from the user's secret $\mathbf{S}$. Of these we expect a fraction

$$\frac{\sum_{k=0}^{N} \binom{N}{k} \cdot (1 - B_{cdf}(\text{Threshold}, TW, p_k))}{2^N}$$

to satisfy (3).

For simplicity, let's choose Threshold $= TWp_e$ so the attacker retains 50 % of the points that are a distance e from the true secret. Thus for a cost of $2^N/\binom{N}{e}$ response evaluations the attacker will end up with

$$\left(\sum_{k=0}^{N} \binom{N}{k} \cdot (1 - B_{cdf}(TWp_e, TW, p_k)) \right) \Big/ \binom{N}{e}$$

secrets with a 50 % probability that one of them is a distance e from the true secret. Every doubling the number of points examined reduces the probability that he misses the true secret by a factor of 2. Taking $2^{N+Q}/\binom{N}{e}$ evaluations improves the chances that includes at least one point within distance e of the true secret to $1-(1-0.5)^Q$.

4.3 Once Close, It's Easy to Get Closer

Suppose we have $\mathbf{S}'$ such that $|\mathbf{S}-\mathbf{S}'|=e$, where e is small. How do we now find $\mathbf{S}$? First consider the N secrets that are distance 1 from $\mathbf{S}'$, *i.e.* consider $\mathbf{S}''$ such that $|\mathbf{S}'-\mathbf{S}''|=1$. For e of these we will have

$$|\mathbf{S}-\mathbf{S}''|=e-1,$$

and for the remaining $N-e$ we have

$$|\mathbf{S}-\mathbf{S}''|=e+1.$$

That is, each distance-1 neighbor of $\mathbf{S}'$ is either a distance-$(e-1)$ or a distance-$(e+1)$ neighbor of the true secret $\mathbf{S}$.

Of course the attacker doesn't know which $N-e$ of the N neighbors are closer, and which e are further away. But he does know that the responses $\mathbf{\Gamma}'$ that are closest to $\mathbf{\Gamma}$ are more likely to come from the distance-$(e-1)$ neighbors. This is made explicit in Figure 3 where we plot

$$Pr\{\mathrm{Simm}(\mathbf{\Gamma},\mathbf{\Gamma}')=d \mid |\mathbf{S}-\mathbf{S}''|=e+1\}=B_{pdf}(d,TW,p_{e+1})$$

and

$$Pr\{\mathrm{Simm}(\mathbf{\Gamma},\mathbf{\Gamma}')=d \mid |\mathbf{S}-\mathbf{S}''|=e-1\}=B_{pdf}(d,TW,p_{e-1})$$

for the scheme as in Figure 2 (*i.e.* $N=80, U=10$ and $k=2$) and $e=12$. Thus, while the tails of the binomials overlap, it is more likely, for example, that if $\mathrm{Simm}(\mathbf{\Gamma},\mathbf{\Gamma}')\geq 95$ that $\mathbf{S}'$ is a distance-$(e-1)$ neighbor of $\mathbf{S}$.

Suppose $\mathbf{S}'$ is a distance-e neighbor of $\mathbf{S}$. Now, if the attacker retains (among the N distance-1 neighbors of $\mathbf{S}'$) the three that maximize $\mathrm{Simm}(\mathbf{\Gamma},\mathbf{\Gamma}'')$ it is overwhelmingly probable that he retains at least one distance-$(e-1)$ neighbor of $\mathbf{S}$. We quantify this in Figure 4 which shows the probability of one of the e secrets that are closer to $\mathbf{S}$ being among the three that maximize $\mathrm{Simm}(\mathbf{\Gamma},\mathbf{\Gamma}'')$. As can be seen for small enough e the attacker is almost guaranteed to have at least one point that is closer among the top three. For example, using the $W=20$ login plot of Figure 4 if $|\mathbf{S}-\mathbf{S}'|=17$ and we choose the three distance-1 neighbors of $\mathbf{S}'$ that have responses closest to $\mathbf{\Gamma}$ the probability is 0.997 that one of them is closer to the true secret (*i.e.* for one of them $\mathbf{S}''$ $|\mathbf{S}-\mathbf{S}''|=16$). Thus given a secret that is close, the attacker has a high probability of getting one point (among three) that is closer still. Since, this probability increases as e decreases, the closer he gets the easier it gets. Now the attacker need merely iterate.

At worst, this would give the attacker 3^e points to consider before finding the secret, which is of course a huge improvement over $\binom{N}{e}$. In fact we can do better; rather than retain 3^m points for each $m = 0, 1, \cdots e$ we retain only the 10 points that maximize $\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{''})$. Figure 5 shows the probability that this simplification finds the correct secret as a function of e. To calculate this curve we measured the number of times that, starting at $\mathbf{S}^{'}$, this algorithm found it's way to $\mathbf{S}$ given that $|\mathbf{S} - \mathbf{S}^{'}| = e$. This was done numerically, by generating random N-bit secrets, and challenges that use a randomly chosen U bits of the secret. The k-bit symbols were generated uniformly at random from the U challenge bits, and we repeat TW times to simulate a stream of W logins. As can be seen the attacker can find his way "home" by, starting at $\mathbf{S}^{'}$, retaining the 10 secrets that maximize $\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{''})$.

4.4 Putting the Pieces Together

We put the pieces of the above analysis together to form a generic brute-force attack to reveal the user's secret $\mathbf{S}$ given the observation from W logins $\mathbf{\Gamma}$. Essentially the attacker chooses enough brute force points to ensure that several of them are close and applies the test of (3) to retain only those that are probably close. On all of the retained points he attempts to iterate and get closer. Those points that actually are close will converge to the true secret. This leads to the following algorithm.

- foreach($2^{N+Q}/\binom{N}{e}$ random secrets $\mathbf{S}^{'}$){
- for ($m = 0, 1, \cdots, e-1$){
- Calculate $\text{Simm}(\mathbf{\Gamma}\mathbf{\Gamma}^{''})$ of the distance-1 neighbors of each element in list
- Retain the 10 secrets that maximize $\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{''})$.
- if ($\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{''}) = TW$ for any list secret) break }}

Since we keep 10 secrets in the list, and each secret has N distance-1 neighbors this algorithm requires $10Ne$ evaluations.

Our fundamental unit of complexity is an evaluation of $\mathbf{\Gamma}^{'}$ for a given secret $\mathbf{S}^{'}$. The overall complexity is the cost of the brute-force search plus the cost of finding the true secret from the points that survive the threshold test:

$$\left(2^N + 10 \cdot N \cdot e \cdot \sum_{k=0}^{N} \binom{N}{k} \cdot (1 - B_{cdf}(TWp_e, TW, p_k))\right) \Big/ \binom{N}{e}. \quad (4)$$

The complexity is inversely related to e. The brute-force (left-hand) term dominates for small e. For example, if $e = N/2$ even a small collection of randomly chosen secrets will contain one e-neighbor, while if $e = 1$ we must include almost the whole secret space.

Choosing the largest possible e for a given scheme will minimize the attacker's complexity. Of course, the attacker must limit his choice of e to those that allow reliable zooming in on the true secret once a distance-e neighbor has been found. For example, in the scheme shown in Figure 5 at $e = 17$ the attacker is almost

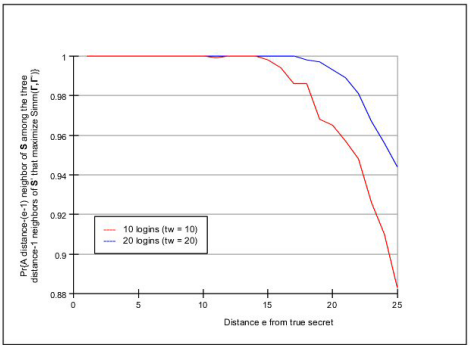

Fig. 4. Given a secret $\mathbf{S}'$ that is distance e from the true secret $\mathbf{S}$ how easy is it to get closer still? The graph shows the probability that the attacker finds a secret distance $e-1$ from $\mathbf{S}$ if he chooses the three distance-1 neighbors of $\mathbf{S}'$ that produce responses most like the observed response. The scheme shown is for $N = 73, U = 8.8, k = 2$ with 10 and 20 logins (left and right resp).

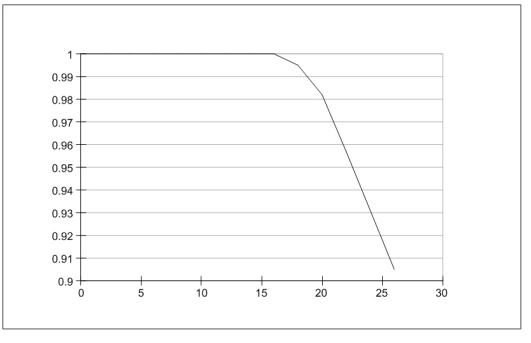

Fig. 5. Given a secret $\mathbf{S}'$ that is distance e from the true secret $\mathbf{S}$ how easy is it to get closer still? The graph shows the probability that by retaining the 10 points that maximize Simm($\mathbf{\Gamma}, \mathbf{\Gamma}''$) and iterating that we find our way from $\mathbf{S}'$ to the true secret. This reduces the complexity from 3^e to $10 \cdot N \cdot e$ points that must be searched. The scheme shown is for $N = 73, U = 8.8, k = 2$ with 10 logins.

guaranteed to successively find secrets that are $16, 15, \cdots 2, 1$ from the true secret, while at $e = 35$ this is extremely unlikely to happen.

We evaluate numerically the largest e that gives the algorithm of Section 4.3 a 0.975 probability of converging to the true secret from a distance e. The results are tabulated in Table 1. As can be seen when $U = 5$ the attacker can start quite far away (*i.e.* e large) and still find the secret, while for $U = 10$ he must start a great deal closer. The table also makes clear that the more logins the attackers observes the easier his task gets (*i.e.* as W increases so also does the value of e from which he can reliably expect to find the secret).

Table 1. The largest value of distance e to have the algorithm of Section 4.3 find the true secret with probability 0.975. We evaluate this for $U = 5$ (left) and $U = 10$ (right). Clearly, the more logins W the attacker observes the easier fidning the true secret becomes. Also, for small U almost any starting point will allows convergence to $\mathbf{S}^{'}$. But even at $U = 4$ a human will require at least 60 seconds to calculate the response to a challenge.

W	$N = 50$	$N = 60$	$N = 70$	$N = 80$
10	19	21	26	39
15	20	29	33	39
20	23	29	34	39
25	24	29	34	39

W	$N = 50$	$N = 60$	$N = 70$	$N = 80$
10	10	11	15	16
15	11	14	16	17
20	13	15	17	18
25	14	16	18	20

Complexity of the User's Task. In Section 4.1 we showed that secrets that are close produce responses that are close. We used the fact that if none of the e bits where $\mathbf{S}$ and $\mathbf{S}^{'}$ differ is among the U bits used to calculate a particular output symbol then the two secrets produce the same output symbol $R(t)$. However, if any of the e bits where they differ was involved we assumed that all outputs were equally likely. So the output would be the same with probability $1/2^k$. Thus, for secrets that differ from $\mathbf{S}$ in any of the U bits used to calculate $R(t)$ the output symbol $R^{'}(t)$ has a uniform random distribution. This implies that each of the k bits of the output symbol depends on all U input bits, but is independent of the other $k-1$ output bits. Hence at least $k(U-1)$ binary decisions must be performed to calculate this symbol. This is in fact a loose lower bound, but tells us that the user must perform at least $M(U-1)$ binary decisions per login.

If the output symbol does not change uniformly at random based on the U input bits things only get better for the attacker. Suppose that only one bit where $\mathbf{S}$ an $\mathbf{S}^{'}$ differ is used among the U that are used to calculate $R(t)$. Now if the probability that the symbol is unchanged is higher than a uniform assignment it merely serves to make the attacker's task easier, and the algorithm of Section 4.4 work better. Previously the attacker could infer closeness only when none of the e differing bits were involved. But now, when only one bit is involved the probability that $R(t) = R^{'}(t)$ is higher than when e bits are involved. Thus the probability that $|\mathbf{S} - \mathbf{S}^{'}|$ is small when

What is Needed to Resist Brute-Force? Since $M(U-1)$ is lower bound on the number of binary decisions the user must perform, we can decide the maximum permissible U for a given burden on the user. Unfortunately this is extremely low. If we assume the user can perform a single binary decision per second then, if a one minute login procedure is acceptable (*i.e.* it would take the user this time to respond to the challenge) then we have $U = 60/20 + 1 = 4$. Even this assumes that the user can reliably perform 60 binary decisions without error. Of course we cannot reduce $M = 20$, since we require a minimum of a 20-bit login.

We summarize the cost of brute-forcing a $U = 5$ scheme in Table 2. For each secret size N, and number of observed logins W we take the value of e from

Table 2. Time in minutes to brute-force a Challenge Response scheme for a given secret size and number of logins observed when $U = 5$ bits of the secret are involved in each output bit. This requires at least 80 binary decisions be made by the user, and a more than one minute login procedure.

W	$N = 50$	$N = 60$	$N = 70$	$N = 80$
10	9.9	24	16	58
15	10.5	15.9	23	32
20	12.2	20.5	30.2	42
25	17.5	27.8	41.4	57

Table 1 and calculate the complexity from (4). We assume that the attacker can perform 1000 evaluations per second for the $N = 50, U = 5, k = 2$ case when he has observed $W = 10$ logins. The costs are given in hours to have a 0.9375 probability of finding the secret.

We summarize the cost of brute-forcing such a scheme in Table 2. The costs are given in hours required to have a 0.9375 probability of finding the true secret using our brute force method. That is, by choosing the largest value of e for a given scheme we calculated the number of trials required using (4). We assumed estimated that 10000 trials per second could be performed. The table makes clear the necessity of using large secrets. We regard it as infeasible to expect the user to perform more than 60 binary decisions for a login, and thus secrets larger than $N = 80$ (*i.e.* the equivalent of a 24 digit PIN) must be used.

5 Conclusion

We have examined the question of whether "Something You Know" can be saved as the sole factor for authenticating a user in the presence of spyware. Our conclusion is negative. We find that in a challenge response scheme the number of bits U of the secret involved in each bit of the response is the key parameter to surviving brute force. Unfortunately the amount of binary decisions the user must perform increases at least linearly with this parameter. This gives a fundamental tradeoff for which there appear to be no good choices. The Weinshall scheme [17], which used about 7.8 bits of the secret per output bit required about 3 minutes for the user to respond to the challenge. If we try to limit the login to a one minute login procedure we find that given enough observed logins the scheme is quickly brute-forced. This is true independent of the details of the scheme.

Golle and Wagner [7] conclude that "something you know" schemes be tested with SatSolver. We would add that measuring the number of bits of the secret involved in each output bit is paramount. Unless U can be made large, brute-forcing is trivial. This suggests that good alternatives between passwords, which do withstand replay, and one-time password or two-factor schemes ar very hard to find.

References

1. http://www.rsasecurity.com
2. Herley, C., Florêncio, D.: How To Login From an Internet Café without Worrying about Keyloggers. In: Symp. on Usable Privacy and Security (2006)
3. Cheswick, W.: Johnny Can Obfuscate: Beyond Mother's Maiden Name. In: Proc. Usenix HotSec (2006)
4. Florêncio, D., Herley, C.: One-Time Password Access to Any Server Without Changing the Server. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 401–420. Springer, Heidelberg (2008)
5. Florêncio, D., Herley, C.: KLASSP: Entering Passwords on a Spyware Infected Machine. In: Jesshope, C., Egan, C. (eds.) ACSAC 2006. LNCS, vol. 4186. Springer, Heidelberg (2006)
6. Florêncio, D., Herley, C., Coskun, B.: Do Strong Web Passwords Accomplish Anything? In: Proc. Usenix Hot Topics in Security (2007)
7. Golle, P., Wagner, D.: Cryptanalysis of a Cognitive Authentication Scheme. In: Symp. on Security and Privacy (2007)
8. Haller, N.: The S/KEY One-Time Password System. In: Proc. ISOC Symposium on Network and Distributed System Security (1994)
9. Herley, C., Florêncio, D.: Phishing as a Tragedy of the Commons. In: NSPW 2008, Lake Tahoe, CA (2008)
10. Jermyn, I., Mayer, A., Monrose, F., Reiter, M.K., Rubin, A.D.: The Design and Analysis of Graphical Passwords. In: Usenix Security (1999)
11. Lamport, L.: Password Authentication with Insecure Communication. Communications of the ACM (1981)
12. Lei, M., Xiao, Y., Vrbsky, S., Li, C.-C., Liu, L.: A Virtual Password Scheme to Protect Passwords. In: Proceedings of IEEE ICC (2008)
13. Lim, J.: Defeat spyware with anti-screen capture technology using visual persistence. In: SOUPS (2007)
14. Pashalidis, A., Mitchell, C.J.: Impostor: A single sign-on system for use from untrusted devices. In: Proceedings of IEEE Globecom (2004)
15. Pering, T., Sundar, M., Light, J., Want, R.: Photographic Authentication through Untrusted Terminals. IEEE Security and Privacy (2003)
16. Suo, X., Zuo, Y., Owen, G.S.: Graphical Passwords: a Survey. In: ACSAC (2005)
17. Weinshall, D.: Cognitive Authentication Schemes Safe Against Spyware. In: Symp. on Security and Privacy (2006)

A Breaking the Virtual Passwords Scheme of Lin *et al.* [12]

The recently suggested Virtual Passwords scheme of Lei *et al.* [12] unfortunately appears to fall to the divide-and-conquer attack described above. In that work, user secret is an n-digit PIN, such that $\mathbf{S} = S_0S_1...S_{n-1}$ where $S_i \in \{0, 1, .., 9\}$. For each login, server presents an n-digit challenge such that $\mathbf{C} = C_0C_1...C_{n-1}$ where $C_i \in \{0, 1, .., 9\}$. The response calculation function is defined as follows:

$$R(t) = \begin{cases} (aS_0 + C_0 + S_1 + b) \bmod Z & i = 0 \\ (aR(t-1) + C_i + S_i + b + S_{i+1}) \bmod Z & i = 1, 2, .., n-1 \end{cases} \quad (5)$$

where a and b are two other random numbers that user has to remember and a is relatively prime to Z in order to make the response function bijective.

In general, since $\mathbf{R}$ and $\mathbf{C}$ are observable by the attacker, the only unknown parameters for each response $R(t)$, are a, b, S_i and S_{i+1}. For the sake of usability, the response for each login is also an n-digit number , such that $R(t) \in \{0, 1, .., 9\}$. Therefore one has to set $Z = 10$, which also implies that $a \in \{1, 3, 7, 9\}$ due to relatively prime constraint. On the other hand, regardless of the actual b value, without loss of generality we can consider that $b \in \{0, 1, .., 9\}$ due to the properties of modular arithmetic. Therefore for each $R(t)$, attacker has to try $4 \times 10 \times 10 \times 10 = 4000$ different combinations of the parameters a, b, S_i and S_{i+1} to see which combination matches the observed response $R(t)$. After one login, the attacker is left only with $4000/10 = 400$ combinations as $R(t)$ can be only one of the 10 possible values. Similarly after second and third login, the attacker will have only 40 and 4 possible choices respectively. And finally, the attacker will get the true parameter combination after observing the fourth login.

In summary, in the worst case attacker reveals $\mathbf{S}$, a and b after observing four login sessions at a cost of $n(4000 + 400 + 40 + 4)$ trials. However, in fact she can do much better both in terms of the number of logins observed and the number of trials, since the same a and b is used for every $R(t)$ and a single $S(i)$ is used for multiple $R(t)$s.

B Explanatory Tables

Password Authentication
Client → Server: $\mathbf{U}, \mathbf{P}$
OTP Authentication
Client → Server: $\mathbf{U}, \mathbf{P}_i$
Challenge Response Authentication
Client → Server: $\mathbf{U}$
Client ← Server: $\mathbf{C}_i$
Client → Server: $f(\mathbf{S}, \mathbf{C}_i)$

Fig. 6. The basic types of access control discussed: password, one-time password and challenge response

$\mathbf{S}$	N-bit secret shared by client and server
$\mathbf{R}$	M-bit client response for a single login
$R(t)$	k-bit symbol of the response ($\mathbf{R} = R(0)R(1)\cdots R(T-1)$)
$f()$	calculation performed by the user
$f_t()$	$R(t) = f_t(\mathbf{S}, \mathbf{C}_i)$
$\mathbf{\Gamma}$	Observed series of W logins ($\mathbf{\Gamma} = \mathbf{R}_0\mathbf{R}_1\cdots\mathbf{R}_{W-1}$)
U	# bits of $\mathbf{S}$ used to calculate each $R(t)$
$B_{pdf}(d, n, p)$	Binomial pdf for n trials with probability p.
$B_{cdf}(d, n, p)$	Binomial cdf for n trials with probability p.

Fig. 7. Summary of notation and symbols used