A New Human Identification Protocol and Coppersmith's Baby-Step Giant-Step Algorithm*

Hassan Jameel Asghar[1], Josef Pieprzyk[1], and Huaxiong Wang[1,2]

[1] Center for Advanced Computing, Algorithms and Cryptography, Department of Computing, Faculty of Science, Macquarie University, Sydney, NSW 2109, Australia
{hasghar, josef, hwang}@science.mq.edu.au,
[2] Division of Mathematical Sciences, School of Physical & Mathematical Sciences, Nanyang Technological University, 50 Nanyang Avenue, 639798, Singapore
{hxwang}@ntu.edu.sg

Abstract. We propose a new protocol providing cryptographically secure authentication to unaided humans against passive adversaries. We also propose a new generic passive attack on human identification protocols. The attack is an application of Coppersmith's baby-step giant-step algorithm on human identification protcols. Under this attack, the achievable security of some of the best candidates for human identification protocols in the literature is further reduced. We show that our protocol preserves similar usability while achieves better security than these protocols. A comprehensive security analysis is provided which suggests parameters guaranteeing desired levels of security.

Keywords: Human Computer Cryptography; Human Identification Protocols; Entity Authenticaion.

1 Introduction

Secure human identification protocols are a form of user authentication protocols through which the human user proves his/her identity to a remote computer server using an insecure public terminal and through an insecure channel. Such protocols are based on shared-key cryptography and are potentially more secure than traditional authentication methods such as passwords, biometrics, tokens or combinations of them, since the adversary has more powers in the threat model. The adversary can view the alphanumerics entered by the user, see the computations done at the terminal and above all observe the information exchanged between the user and the remote server during a protocol session. Much stronger models allow the adversary to actively interfere with the communication channel. This scenario was conceived by Matsumoto and Imai in [4].

* This is the full version of the paper with the same title in J. Zhou and M. Yung (Eds.): Applied Cryptography and Network Security, ACNS 2010, LNCS 6123, Springer, 2010.

Although, human identification protocols secure against active attacks is coveted, it is extremely hard to construct one that ensures both acceptable security and good human executability. To date there have been a handful of proposals known to resist some active attacks [1, 6, 8, 14]. Among them only the *sum of k mins* protocol proposed in [1] has been constructed to be secure against generic active adversaries; yet, the protocol falls short of usability. The rest of the protocols only treat security against a known set of active attacks. Due to the difficulty of constructing usable protocols secure against active attacks, recently the focus of the research community has been on security against passive (eavesdropping) adversaries [2, 3, 10, 12, 13]. Despite this being a weaker threat model, there is still no widely accepted human identification protocol secure against passive adversaries and this remains an open problem. Recently proposed protocols can be used for only a small number of authentication sessions if a certain level of usability is desired. Loosely speaking, most of these protocols consist of a shared secret that is a set of k objects out of n. The protocol involves a series of challenges from the server and corresponding responses by the user which are constructed as a function of the secret and the challenges. Since the challenges and their responses are communicated in the open, the adversary can always do a search after learning a few challenge-response pairs to find the secret. We shall call protocols belonging to this general category as *k-out-of-n protocols*. Hopper and Blum showed a bound on what security is achievable for this class of protocols assuming a certain generic time-memory tradeoff attack to be optimal [1, §6]. In other words, all k-out-of-n protocols are susceptible to this time-memory tradeoff attack, even if they do not have any other weaknesses. This bound severely lessens accomplishable security for small values of k. A smaller value of k is necessary to ensure that human memory and computational requirements are low. Despite their findings, some of the recent proposals have ignored security evaluation against time-memory tradeoff attacks [10, 12].

Hopper and Blum also proposed two protocols for human identification. One of them is the now famous HB protocol based on the problem of learning parity in the presence of noise. However, the variant of this protocol proposed as a human identification protocol uses a small hamming weight k. We call this protocol the k-weight HB protocol to distinguish it from the HB family of protocols used for pervasive devices. The other protocol is the sum of k mins protocol mentioned before. We use the variant that is constructed to be secure against passive adversaries only. The two protocols achieve good security against passive adversaries but fall short of the theoretically achievable bound imposed by the time-memory tradeoff attack. The reason for this is to avoid lack of usability.

In this paper, we propose a protocol that achieves better security than the aforementioned protocols while preserving similar usability. We also propose a new time-memory tradeoff attack on human identification protocols that has lower time-complexity than the one sketched in [1, §6]. The main component of the attack is Coppersmith's baby-step giant-step algorithm which has its application in solving the restricted hamming weight discrete logarithm problem [15]. Once again, this attack can be applied to all k-out-of-n protocols. It performs

better than the attack mentioned in [1] on the two proposed protocols in that paper, further reducing their security. Our protocol shows better security under both time-memory tradeoff attacks. We also rigorously analyze other possible attacks to demonstrate their efficacy, or lack thereof, in breaking our protocol. Our focus is restricted to passive adversaries.

2 Related Work

The first human identification protocol that was constructed to be secure in the aforesaid threat model was proposed by Matsumoto and Imai in [4]. This scheme was shown to be insecure by Wang, Hwang and Tsai in [6] and the authors proposed some improvements but with a severe loss in usability. Matsumoto proposed another scheme in [7] based on techniques from linear algebra, but it can only be used for a small number of authentication sessions. Yang and Teng proposed a couple of protocols in [8] which with the suggested memory requirement of three 20 to 40-bit secrets, is surely infeasible for most humans. Hopper and Blum proposed two human identification protocols in [1]. One of them is the k-weight HB protocol and the best known attack on this protocol is the meet-in-the-middle attack [1, §3.1, pp. 58]. The protocol requires the human to toss a coin with a fixed probability between 0 and 0.5, which is arguably not achievable without additional aid. The other proposal by the authors is the sum of k mins protocol. The version of this protocol secure against passive adversaries is, in our opinion, the most secure and usable protocol published to date. Our protocol performs slightly better, in that it does not use ordered pairs and hence the user does not necessarily have to remember the secret objects in a specific order. Furthermore, the security of our protocol is much higher for similar parameter values.

Li and Shum [14] describe two protocols which resist a large number of passive attacks and can be used for a large number of authentication sessions. However, the security analysis of their protocols does not take time-memory tradeoff attacks into account. Weinshall [10] proposed a slightly different protocol in which the user has to remember images instead of alphanumeric secrets. However, the main structure of the secret is the same as before; the secret is a small subset of k objects out of n. The protocol was cryptanalyzed and broken by Golle and Wagner in [11]. Use of images as memory aids has been employed previously, such as in [5]. However, straightforward protocols in which the user is required to click on a subset of secret pictures in a set of given pictures is susceptible to shoulder-surfing or passive observer attacks. For a comprehensive survey of these protocols and others see [9]. More recently, the authors of [2] and [3] proposed a slightly different concept in which the internal properties of images are used as secrets. However, the security of the protocols cannot be concretely demonstrated as it relies on some unproven assumptions. Another recent attempt is by Bai et. al. in [12], but their protocol can only be used for a small number of authentications; precisely 10, using the default parameters. A different line of research is to develop special purpose hardware that utilizes other human

senses (such as the sense of touch). Assuming the device to be tamper-proof, this makes the protocol secure against shoulder-surfers. The protocol from [13] is an example.

3 Preliminaries: Definitions and Threat Model

Throughout this text, the prover is denoted by $\mathcal{H}$ and the verifier by $\mathcal{C}$. This is to acknowledge that in the real world, the prover will be a human user and the verifier will be a remote computer (server). The goal of an identification protocol is to authenticate $\mathcal{H}$ to $\mathcal{C}$[3]. We restate the definitions of identification protocols and human executable protocols from [1] for reference. A protocol is defined as a sequence of interactions between a pair of public and probabilistic interactive turing machines (ITMs) $(\mathcal{H}, \mathcal{C})$. The result of the interaction between these two (ITMs) with respective inputs x and y is denoted by $\langle \mathcal{H}(x), \mathcal{C}(y) \rangle$. The transcripts of bits exchanged between $\mathcal{H}$ and $\mathcal{C}$ during this interaction is denoted by $T\left(\mathcal{H}(x), \mathcal{C}(y)\right)$.

Definition 1. *An* identification protocol *is a pair of public, probabilistic interactive turing machinces (ITMs) $(\mathcal{H}, \mathcal{C})$ with shared auxiliary input z, such that the following conditions hold:*
– *For all auxiliary inputs z,* $\Pr\left[\langle \mathcal{H}\left(z\right), \mathcal{C}\left(z\right) \rangle = \mathsf{accept}\right] > p_0$
– *For each pair $x \neq y$,* $\Pr\left[\langle \mathcal{H}\left(x\right), \mathcal{C}\left(y\right) \rangle = \mathsf{accept}\right] < 1 - p_0$ *where* $0.5 < p_0 \leq 1$.

When $\langle \mathcal{H}, \mathcal{C} \rangle = \mathsf{accept}$, we say that $\mathcal{H}$ authenticates to $\mathcal{C}$. The above definition also takes those protocols into account which even reject legitimate provers, albeit with a small probability. An example is the Hopper and Blum (HB) protocol proposed in [1]. For human computational ability, we shall use the following definition from [1], which states that the computations done by the probabilistic turing machine $\mathcal{H}$ should be easy enough to be carried out by a human.

Definition 2. *An identification protocol $(\mathcal{H}, \mathcal{C})$ is* (α, β, τ)-human executable *if at least a $(1 - \alpha)$ portion of the human population can perform the calculations $\mathcal{H}$ unaided and without errors in at most τ seconds with probability greater than $(1 - \beta)$.*

The goal is to minimize α, β and τ. Concrete values to these parameters can be assigned by either an intuitive approximation or where it is hard to do so, actual user experiments can be carried out [1]. As an exemplary use of these definitions, we see that the traditional password-based authentication system satisfies the definition of an identification protocol, since if $\mathcal{H}$ and $\mathcal{C}$ share the same password, then $\mathcal{H}$ authenticates to $\mathcal{C}$ with probability 1. Otherwise, $\mathcal{H}$ is accepted with probability 0. Also, we can conjecture that it is $(\approx 0, \approx 0, 5)$-human executable for say a length 10 password, based on our everyday experience.

[3] While mutual authentication is desirable, in general it is much harder to construct such a protocol. Partly due to the inability of humans to construct truly random challenges.

3.1 Security Definitions

The adversary, denoted by $\mathcal{A}$, has passive access to the channel between $\mathcal{H}$ and $\mathcal{C}$. In this light, the adversary views both $\mathcal{H}$ and $\mathcal{C}$ as oracles. The following definition treats $\mathcal{H}$ and $\mathcal{C}$ as Interactive Turing Machines. In the identification protocols discussed in this paper the computations done by $\mathcal{H}$ have to be carried out by a human user. Therefore, all such computations should be done by the human mentally, or else any security proved against $\mathcal{A}$ will be superfluous.

The following security definition is taken from [1] and involves the passive adversary $\mathcal{A}$ defined above.

Definition 3. *An identification protocol $(\mathcal{H}, \mathcal{C})$ is (p, m) secure against passive adversaries if for all computationally bounded adversaries $\mathcal{A}$ and for all auxiliary inputs z,*

$$\Pr\left[\langle \mathcal{A}\left(T^m\left(\mathcal{H}\left(z\right), \mathcal{C}\left(z\right)\right)\right), \mathcal{C}\left(z\right)\rangle = \mathsf{accept}\right] \leq p$$

Here $T^m\left(.,.\right)$ represents the transcript of m independent communication sessions between $\mathcal{H}$ and $\mathcal{C}$.

If $\langle \mathcal{A}(T^m(\mathcal{H}, \mathcal{C})), \mathcal{C}\rangle = \mathsf{accept}$ for some m, we say that $\mathcal{A}$ impersonates $\mathcal{H}$. In some identification protocols, if less than a threshold number of communication sessions are observed, even information theoretic security is achievable. However, this threshold is generally too low and hence information theoretic security can only be guaranteed for a handful of communication sessions. As a toy example, consider a protocol in which $\mathcal{H}$ and $\mathcal{C}$ share an ordered pair of letters selected uniformly at random from the lower case English alphabet as a secret. Suppose the secret is (a, b). On a communication session, $\mathcal{C}$ sends $\mathcal{H}$ one of the following challenges: $(1, 0), (1, 1), (0, 1)$. If $\mathcal{C}$ prompts for $(0, 1)$, the reply from $\mathcal{H}$ is b. Thus, after observing this communication session, even a computationally unbounded adversary can not impersonate $\mathcal{H}$ with probability better than $1/26$, as it does not know the other letter.

We define a challenge-response identification protocol as the following sequence of messages communicated between $\mathcal{H}$ and $\mathcal{C}$:

$$\mathsf{request}, (c_1, r_1), (c_2, r_2), \ldots, (c_m, r_m), \mathsf{accept/reject}$$

The message request is sent by $\mathcal{H}$ to $\mathcal{C}$. It symbolizes a request to start a protocol and contains information about $\mathcal{H}$ such as its identity. Each pair (c_i, r_i) consists of a challenge c_i drawn from some *challenge space* by $\mathcal{C}$ and a response r_i composed from a *response space* by $\mathcal{H}$. At the end of m challenge-response pairs, $\mathcal{C}$ sends the message accept or reject. We call this sequence of messages as one *session* (or an authentication session) of the challenge-response identification protocol.

4 Proposed Protocol

Notation We refer to a vector of n elements or an n-tuple as an ordered list of n elements. If $\mathbf{c}$ is a vector of n-elements, then $\mathbf{c}[i]$ denotes the ith element of $\mathbf{c}$,

for $0 \leq i \leq n-1$; the index i is called the ith location in $\mathbf{c}$. Let n be such that $n = ab$ for positive integers a and b. We call a the *jump constant* for reasons that will be clear later. Let k and d be positive integers. Let $\mathbf{c}$ be a vector of n integers drawn uniformly at random from the set $\{0, 1, \ldots, d-1\}$.

We first describe the protocol formally and then show an implementation that is human friendly. Sections 4.1 and 4.2 give a less technical description of the protocol with example values for the parameters.

Protocol 1.

Setup: Let n, a, b, m, k and d be public parameters, with $n = ab$. $\mathcal{C}$ and $\mathcal{H}$ choose $k+1$ locations in $\mathbf{c}$. These locations are essentially a set of integers: $s_0, s_1, \ldots, s_k$, where $0 \leq s_i \leq n-1$. s_0 is called the starting location. All these locations constitute the secret.

1: **repeat**
2: $\mathcal{C}$ updates $m \leftarrow m-1$, generates the vector $\mathbf{c}$ and sends it to $\mathcal{H}$.
3: $\mathcal{H}$ assigns $t \leftarrow s_0$.
4: $\mathcal{H}$ updates $t \leftarrow (t + a \cdot \mathbf{c}[s_0]) \bmod n$.
5: **for** $1 \leq i \leq k$ **do**
6: $\mathcal{H}$ updates $t \leftarrow (t + \mathbf{c}[s_i]) \bmod n$.
7: $\mathcal{H}$ sends $\mathbf{c}[t]$ to $\mathcal{C}$.
8: **if** the answer is incorrect **then**
9: $\mathcal{C}$ outputs reject.
10: **until** $m = 0$
11: $\mathcal{C}$ outputs accept.

Lemma 1. *Let $\mathfrak{S}_1$ and $\mathfrak{S}_2$ be two sets of secret locations in $\mathbf{c}$. Then, Protocol 1 is an identification protocol with:*

$$\Pr\left[\langle \mathcal{H}\left(\mathfrak{S}_1\right), \mathcal{C}\left(\mathfrak{S}_2\right)\rangle = \mathsf{accept}\right] \leq d^{-m}$$

if $\mathfrak{S}_1 \neq \mathfrak{S}_2$ and 1 otherwise.

Proof. Since the protocol is deterministic, if $\mathfrak{S}_1 = \mathfrak{S}_2$, $\mathcal{C}$ will accept $\mathcal{H}$ with probability 1. For $\mathfrak{S}_1 \neq \mathfrak{S}_2$, we see that since each element of $\mathbf{c}$ is generated uniformly at random from the integers $\{0, 1, \ldots, d-1\}$, the probability that two different sets of locations generate the same output for $\mathbf{c}$ is at most $1/d$. The result follows for m iterations. □

4.1 User Friendly Implementations

Both graphical and textual implementations are possible for our protocol. In a graphical implementation, we can use n graphical objects such as software icons. In this case, the user's secret is a set of k icons out of n. Here we illustrate an example test-based implementation.

We represent the secret space, i.e. the locations in $\mathbf{c}$ by an alphabet. For instance, the English alphabet is a candidate. In this subsection, we abuse the

notation a little to denote the human user by $\mathcal{H}$. The vector $\mathbf{c}$ is presented to $\mathcal{H}$ in the form of a grid with $a \times b$ cells, where $b = n/a$ (a is chosen such that it divides n). Each location in $\mathbf{c}$ is mapped to a unique character in the secret space alphabet. Each cell in the grid contains a unique character from the secret space, below which is the corresponding random digit from $\mathbf{c}$. The secret is then a string from this alphabet, instead of a set of integers of vector locations. Thus, the starting location is also a character from this alphabet.

Given a challenge "grid", $\mathcal{H}$ locates the cell containing the starting character. This corresponds to the location s_0 in the formal description. $\mathcal{H}$ then looks at the digit corresponding to this cell. Let the digit be d_0. $\mathcal{H}$ moves d_0 steps vertically downwards, in a circular way, thus reaching a new character location in the same grid. Call this location l_0. $\mathcal{H}$ then looks at the digit in the cell containing s_1. Let d_1 be the digit. It now moves horizontally to the right of the location l_0 and moving to the start of the next row if the end of the row is reached. This results in the new location l_1. $\mathcal{H}$ continues to move horizontally according to the digits corresponding to the rest of its secret locations. If the bottom right corner of the grid is reached, $\mathcal{H}$ moves to the top left corner, thus moving in a cycle. At the end of this procedure, $\mathcal{H}$ simply outputs the digit corresponding to the final character location thus reached. Figure 1 shows an example. Here $a = 12$ and $b = 6$, which implies that $n = 72$. In this example and also through most of the text, we will choose $d = 10$, as this is a common base for humans. The alphabet is composed of the characters a, ..., z, A, ..., Z, 0, ..., 9 and special characters: !, @, #, $, %, ∧, &, ∗, (,). Notice that the order of the characters remains the same for all challenges and only the digits corresponding to these characters change for different challenges. For the graphical implementation, we simply replace the

a	b	c	d	e	f	g	h	i	j	k	l
3	2	6	9	2	1	7	5	4	4	6	8
m	n	o	p	q	r	s	t	u	v	w	x
1	6	7	4	9	7	5	3	2	7	6	1
y	z	A	B	C	D	E	F	G	H	I	J
3	2	5	1	5	2	9	6	6	8	6	0
K	L	M	N	O	P	Q	R	S	T	U	V
3	1	7	4	9	7	5	3	4	7	6	1
W	X	Y	Z	0	1	2	3	4	5	6	7
6	3	8	2	6	8	3	2	9	5	8	0
8	9	!	@	#	$	%	∧	&	∗	(	)
4	7	2	0	9	1	5	3	6	3	4	9

Fig. 1. An example challenge grid.

alphabet with a corresponding set of graphical objects. While the text-based implementation is shown here as an illustration of our protocol, we recommend graphical implementation as it is more user-friendly and has less security issues, such as resistence to dictionary attacks.

4.2 Different Ways of Computation

The above mentioned procedure is one way the human user can perform the protocol steps. There are a number of other ways in which the protocol can be executed. The user can choose any method he or she prefers.

For instance, a different and probably more efficient way is sketched here.

1. Ignore the starting location and add all the digits corresponding to the remaining k secret locations.
2. From the starting location, move d_0 steps vertically downwards (continuing from the top if the bottom of the grid is reached), where d_0 is the digit corresponding to the starting location.
3. Divide the sum obtained in Step 1 by the jump constant a to get a quotient and a remainder. The quotient is the number of vertical steps and the remainder is the number of horizontal steps to be taken. The user can then follows these steps from the location reached in Step 2 and finally output the digit corresponding to the location thus reached.

If the jump constant a is a multiple of 10 then the above division can be performed by most humans mentally. Thus to make this method easy for most users, n and a can be chosen to be 200 and 20 respectively. These parameters are easy for humans to use. k can be chosen somewhere between 10 and 15.

5 Security Analysis

Recall that for security, we consider the computationally bounded passive adversary of Definition 3. The adversary can view every challenge-response pair. The adversary knows the description of Protocol 1 and the public parameters specified in that protocol. The only thing hidden from the adversary is the set of secret locations shared by $\mathcal{C}$ and $\mathcal{H}$. In this section, we assume that the calculations $\mathcal{H}$ can be performed by a human mentally without any additional aid and thus $\mathcal{A}$ gains no advantage in observing the human user's behavior. We also assume that the secret locations are chosen uniformly at random from the set of all possible secret locations.

Given m challenge-response pairs $(\mathbf{c}_1, r_1), \ldots, (\mathbf{c}_m, r_m)$, the goal of $\mathcal{A}$ is to impersonate $\mathcal{H}$ either by partially or completely learning the secret locations or by impersonating $\mathcal{H}$ by guessing the answers without knowledge of the secret. We assume that all these challenge-response pairs correspond to successful authentication sessions between $\mathcal{H}$ and $\mathcal{C}$. We first look at some obvious attacks following which we shall show more sophisticated attacks.

5.1 Some Obvious Attacks

Random Guess 1 The most obvious impersonation attack is to randomly guess the answers when prompted for a challenge. Since the output is in the range $\{0, 1, \ldots, d-1\}$, a random guess from this set will be successful with probability $1/d$ for each challenge. For m challenges, this probability is d^{-m}.

Random Guess 2 Another method for impersonation is to guess the secret and then answer a challenge by following the steps of Protocol 1 correctly. We see that there are a total of n^{k+1} possible ways of choosing $k+1$ locations in $\mathbf{c}$. However, due to the commutativity of the addition operation, any permutation of a given set of locations will generate the same output. Thus we are looking at the number of ways of choosing the starting location times the number of ways of choosing k locations from a set of n locations with replacement and without order, which is exactly: $n\binom{n+k-1}{k}$. Thus the probability of success in guessing the correct secret is $(n\binom{n+k-1}{k})^{-1}$.

The hitherto mentioned attacks are online attacks and effective measures exist to prevent them. For instance, if there are more than a threshold number (say 3) of unsuccessful consecutive login attempts, the user account can be blocked.

Brute Force The bruteforce attack has complexity $O(n\binom{n+k-1}{k})$. It works by trying all possible secret locations satisfying m challenge-response pairs. In the next section, we shall see that after observing m challenge-response pairs, the expected number of candidates for the secret is: $n\binom{n+k-1}{k}/d^m$. To reduce this number to a unique secret, $\mathcal{A}$ needs on average: $n\binom{n+k-1}{k}/d^m \approx 1 \Rightarrow m \approx \log_d(n\binom{n+k-1}{k})$ challenge-response pairs. Thus $O(\log_d(n\binom{n+k-1}{k}))$ challenge-response pairs are enough to find the secret uniquely. With a lower number of challenge-response pairs, there are multiple candidates and even a computationally unbounded adversary cannot distinguish between them. For concrete values, we see that if $n = 200$ and $k = 15$, the brute force attack has complexity roughly 2^{83}. An attack with this complexity is generally considered intractable.

5.2 Algebraic Interpretation

Given m challenge-response pairs $(\mathbf{c}_1, r_1), \ldots, (\mathbf{c}_m, r_m)$, we now consider the problem of finding the secret locations $s_0, \ldots, s_k$. We attempt to describe this problem algebraically. The following notations will be used henceforward: If $\mathfrak{A}$ is a set, then $|\mathfrak{A}|$ denotes the number of elements in $\mathfrak{A}$. If x and y are two strings, then $x||y$ represents their concatenation. If x and y are two integers, $[x, y]$ represents the interval of integers between x and y inclusive. A string of n-elements can be transformed into a vector of n-elements in a natural way, and the two terms will be used interchangeably in the following.

For any challenge-response pair $(\mathbf{c}, r)$, a location $\hat{r}$ satisfying $\mathbf{c}[\hat{r}] = r$ is called a *satisfying location* for $(\mathbf{c}, r)$. For $1 \leq i \leq m$, let $\mathfrak{R}_i$ be the set of all satisfying locations for $(\mathbf{c}_i, r_i)$. Let $\mathfrak{R}^m = \mathfrak{R}_1 \times \cdots \times \mathfrak{R}_m$ be the m-ary cartesian product over these m sets. We represent the elements of $\mathfrak{R}^m$ as m-element vectors, $\hat{\mathbf{r}} = [\hat{r}_1 \cdots \hat{r}_m]^T$, in an obvious way. For any vector $\mathbf{x}$, let $|\mathbf{x}|$ denote the number of elements in $\mathbf{x}$. Define the weight of $\mathbf{x}$ as:

$$\mathsf{wt}(\mathbf{x}) = \sum_{i=1}^{|\mathbf{x}|} x_i$$

where $x_i = \mathbf{x}[i]$, the ith element of $\mathbf{x}$. Define the matrix C as:

$$C = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{m,1} & c_{m,2} & \cdots & c_{m,n} \end{bmatrix}$$

where $c_{i,j} = \mathbf{c}_i[j]$. Then, for each $\hat{\mathbf{r}} \in \mathfrak{R}^m$ we have:

$$\begin{bmatrix} aC\ C \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \hat{\mathbf{r}} \bmod n \tag{1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are n-element vectors with $\mathsf{wt}(\mathbf{x}) = 1$ and $\mathsf{wt}(\mathbf{y}) = k$. Clearly, $\mathbf{x}$ corresponds to the starting secret location and $\mathbf{y}$ corresponds to the k remaining secret locations. Notice that $\mathbf{y}$ need not be a binary vector since each location can be chosen more than once. There are $|\mathfrak{R}^m|$ such equations and we cannot write this as a system of fewer equations as for each satisfying location $\hat{r}$ for a challenge-response pair $(\mathbf{c}, r)$, r is independent of $\hat{r}$. This is true since each element of $\mathbf{c}$ is generated uniformly at random from $\{0, 1, \ldots, d-1\}$.

We first estimate $|\mathfrak{R}^m|$.

Lemma 2. *For $1 \leq i \leq m$, $E[|\mathfrak{R}_i|] = n/d$. And $E[|\mathfrak{R}^m|] = (n/d)^m$.*

Proof. Let $i \in [1, m]$. In each pair $(\mathbf{c}_i, r_i)$, if every element of $\mathbf{c}_i$ is generated uniformly at random from $\{0, 1, \ldots, d\}$, then the expected number of times r_i occurs in $\mathbf{c}_i$ is n/d. This is exactly the expected number of satisfying locations for r_i. The result for $|\mathfrak{R}^m|$ follows from its definition. □

There is exactly one element in $\mathfrak{R}^m$ that contains the satisfying locations corresponding to $\mathcal{H}$'s secret. We denote this by $\hat{\mathbf{r}}_s$. For more compact notation, define $\mathbf{s} = [\mathbf{x}\,\mathbf{y}]^T$ and $C' = [aC\ C]$. Finding a unique $\mathbf{s}$ satisfying $C'\mathbf{s} = \hat{\mathbf{r}}_s \bmod n$ thus translates into finding the secret locations of $\mathcal{H}$. There are $2n$ unknowns in $\mathbf{s}$. If m is small, the number of solutions for $\mathbf{s}$ is very large. On the other hand, higher m means a higher value of $E[|\mathfrak{R}^m|] = (n/d)^m$, which in turn means more equations to be solved, since $\mathcal{A}$ has no way to distinguish $\hat{\mathbf{r}}_s$ from other elements in $\mathfrak{R}^m$.

Let Solve Equations be an algorithm that finds solutions (possibly multiple) to the linear system defined in Equation 1. Further, let $\tau(n, k, m, d)$ be the time complexity of this algorithm. We have the following probabilistic algorithm for finding $\mathbf{s}$.

Algorithm 1.

Input: The matrix C' and the set $\mathfrak{R}^m$.

1: Initialize an empty set $\mathfrak{S}$.
2: **for** each $\hat{\mathbf{r}} \in \mathfrak{R}^m$ **do**
3: Run Solve Equations on input $C'\mathbf{s} = \hat{\mathbf{r}} \bmod n$. If any solutions are found, assign them to the set $\mathfrak{S}$.

4: Output an element uniformly at random from $\mathfrak{S}$.

This algorithm will perform well probabilistically if $|\mathfrak{S}|$ is small. As we have found before, the expected size of $\mathfrak{R}^m$ is $(n/d)^m$. There are a total of n^m possible different output vectors for $C'\mathbf{s} \bmod n$. Therefore, the probability that an $\mathbf{s}$, different from $\mathcal{H}$'s secret, is in the set $\mathfrak{S}$ can be estimated as $\frac{(n/d)^m}{n^m} = 1/d^m$. Thus this probability becomes lower as m increases. Therefore, we can assume that the performance of this algorithm is good. The expected time complexity of Algorithm 1 is:

$$O\left(\tau\left(n,k,m,d\right)\left(\frac{n}{d}\right)^m\right)$$

If gaussian elimination is used as the Solve Equations algorithm, we require $m \geq 2n$, but this means that the expected size of $\mathfrak{R}^m$ will be greater than or equal to $(n/d)^{2n}$. We can see the complexity of this algorithm with concrete values. Let $n = 100$, $d = 10$ and $k = 16$. Then, we have $m \geq 2n = 200$. Gaussian elimination takes time $O(n^3)$ giving a total approximate time 2^{687}, which is surely infeasible.

Since the weights of $\mathbf{x}$ and $\mathbf{y}$ are restricted, we might still be able to use other methods with a smaller value of m. To this end, given m, we find the number of possible solutions of the following equation:

$$C'\mathbf{s} = \hat{\mathbf{r}} \bmod n \tag{2}$$

where $\hat{\mathbf{r}} \in \mathfrak{R}^m$. Since $\mathsf{wt}(\mathbf{x}) = 1$ and $\mathsf{wt}(\mathbf{y}) = k$, with $m = 0$ there are a total of $n\binom{n+k-1}{k}$ possible choices for $\mathbf{s}$. With $m = 1$, we expect a $1/n$ fraction of these choices to satisfy the above equation. Continuing on this way, we see that the expected number of possible choices for $\mathbf{s}$ are:

$$\frac{n\binom{n+k-1}{k}}{n^m} = \frac{\binom{n+k-1}{k}}{n^{m-1}}$$

Since the expected size of $\mathfrak{R}^m$ is $(n/d)^m$, we see that the combined expected number of solutions are:

$$\frac{n\binom{n+k-1}{k}}{d^m}$$

Equating the above expression to 1, we get:

$$m = \log_d n + \log_d \binom{n+k-1}{k} \tag{3}$$

Thus $m = O(\log_d n + \log_d \binom{n+k-1}{k})$ is required on average to find a unique value of $\mathbf{s}$ in Equation 2. By using the concrete values as above, we find that the resulting value of m from Equation 3 is approximately 29. Thus in theory, we can have an algorithm that solves the problem with $m = 29$. However, this value of m implies that $(n/d)^m \approx 2^{96}$. Thus whether or not a Solve Equations algorithm that works for smaller values of m can be found, the overall time complexity of Algorithm 1 is still very high. Thus it is not possible to improve this

complexity without a different approach. Informally speaking, the main reason for this interesting result is that in our protocol, the computations are also done on the location indices and not just the digits corresponding to these locations as in previously proposed protocols. Since the digits are generated uniformly at random, the final answer is not linearly dependent on the location indices. Next, we present a time-space tradeoff algorithm that utilizes fewer challenge-response pairs and has better time complexity.

5.3 Time-Memory Tradeoff

In [1, §6], Hopper and Blum sketched a meet-in-the-middle algorithm which, on k-out-of-n protocols, has average-case time complexity of:

$$O\left(n^{k\left(1-\frac{\ln d}{2\ln Q}\right)}\right) \tag{4}$$

Here Q is an intermediate result, which in our protocol corresponds to the range of the intermediate locations during the computation of the protocol. Thus in our protocol, $Q = n$. Our protocol and the protocols from [1] as well as many other protocols in literature loosely fall in this category of protocols (the only difference in our protocol is that we have a starting location that is computed differently from the k remaining locations). For the k-weight HB protocol, the average-case time complexity of this attack is: $O(\binom{n}{k/2})$ [1, §3.1, pp. 58]. This is true due to two reasons. First the protocol uses the addition operation, which is commutative, and the user has to choose k *unique* locations as a secret. Therefore, the number of possible secrets are $\binom{n}{k}$ instead of n^k. Secondly, Q equals d in their protocol. Similarly, for the sum of k mins protocol the average-case time complexity of this attack is $O(\binom{n(n-1)/2}{k/2})$[1, §3.2, pp. 59]. However, since the size of the secret is exactly twice than in the k-weight HB protocol, the comparative time complexity is $O(\binom{n(n-1)/2}{k/4})$.

This attack is essentially a time-memory tradeoff. The time-memory tradeoff attack that we present here employs a deterministic *baby-step giant-step* algorithm by Coppersmith summarised in [17, pp. 109] and detailed in [15, §2.1]. On k-out-of-n protocols, the resulting attack has average-case time complexity of:

$$O\left(\frac{n^{k\left(1-\frac{\ln d}{2\ln Q}\right)} n^{\frac{\ln d}{\ln Q}}}{2^{\frac{k}{2}\frac{\ln d}{\ln Q}}}\right)$$

which is better than the former if $2^{k/2} < n$ or $k < 2\log_2 n$. The space complexities of the two attacks are the same. While the time complexity is comparable to the previously mentioned meet-in-the-middle attack for generic k-out-of-n protocols, our attack however, performs much better on the two protocols in [1]. The average-case time complexity of our algorithm on the k-weight HB protocol is $O(\binom{n/2}{k/2})$ and on the sum of k mins protocol is $O(\binom{n(n-1)/4}{k/4})$. This is substantially smaller than the previous result.

The original application of Coppersmith's algorithm is to solve the restricted hamming weight discrete logarithm problem [16]. But since this algorithm essentially utilizes the knowledge of the restricted hamming weight, we can modify it to solve our problem. We notice that there are several other deterministic algorithms that perform asymptotically better than Coppersmith's algorithm like the one proposed by Stinson of time complexity $O\left(k^{3/2}(\ln n)\binom{n/2}{k/2}\right)$ [15]. However, for the choice of parameters used in this paper, the performance is comparable to Coppersmith's algorithm if not worse. There are also some probabilistic variants, but which cannot be applied here since the discrete logarithm is always unique and this is not necessarily the case with the candidate locations in our problem for small values of m. For larger values of m time-memory trade-off algorithms become infeasible. We now describe the attack on our protocol and derive its time complexity. The derivations of the other results mentioned above are similar.

For simplicity, we assume n and k to be even integers. For arbitrary n and k, the attack can be carried out with minor differences [15, §5]. For $0 \leq i \leq n-1$, define $\mathbf{b}_i$ to be a vector of length n such that $\mathbf{b}_i[l+1] = 1$ whenever, $l \equiv i + j \bmod n$, for $0 \leq j \leq n/2 - 1$, and 0 otherwise. Clearly, for all i, $\mathbf{b}_i$ is a binary vector with $\mathsf{wt}(\mathbf{b}_i) = n/2$. Let $\mathfrak{B} = \{\mathbf{b}_i : 0 \leq i \leq n/2-1\}$. Now, let $\mathfrak{Y}$ be the set of all n-element vectors. From [15, §2.1], we see that for all $\mathbf{y} \in \mathfrak{Y}$ with $\mathsf{wt}(\mathbf{y}) = k$, there exists a $\mathbf{b} \in \mathfrak{B}$, such that:

$$\mathbf{y} \cdot \mathbf{b} = \frac{k}{2}$$

For any $\mathbf{y}_1, \mathbf{y}_2 \in \mathfrak{Y}$, $\mathbf{y}_2$ is called the *sub* of $\mathbf{y}_1$, denoted $\mathbf{y}_2 \prec \mathbf{y}_1$, if $\mathbf{y}_1[l] = 0 \Rightarrow \mathbf{y}_2[l] = 0$ for $1 \leq l \leq n$. Let $\mathbf{1}$ denote the binary vector of weight n. Let $\mathbf{y}_{=k/2}$ denote a vector whose weight is $k/2$. We divide $\mathbf{s}$ into two parts: $\mathbf{s}_1 = [\mathbf{x}\ \mathbf{y}_{=k/2}]^T$ and $\mathbf{s}_2 = [\mathbf{0}\ \mathbf{y}_{=k/2}]^T$. We assume there to be a hash table, initially empty, which will be used as a data structure in this attack.

Algorithm 2.

Input: The set $\mathfrak{B}$ and m challenge-response pairs.

1: Initialize an empty set $\mathfrak{S}$.
2: **for** $0 \leq i \leq n/2 - 1$ **do**
3: **for** each possible vector $\mathbf{s}_1 = [\mathbf{x}\ \mathbf{y}_{=k/2}]^T$ such that $\mathbf{y}_{=k/2} \prec \mathbf{b}_i$ **do**
4: Compute the string $q \leftarrow \mathbf{c}_1 \cdot \mathbf{s}_1 \bmod n || \cdots || \mathbf{c}_m \cdot \mathbf{s}_1 \bmod n$.
5: Insert this m-digit string q along with $\mathbf{s}_1$ in the hash table.
6: **for** each vector $\mathbf{s}_2 = [\mathbf{0}\ \mathbf{y}_{=k/2}]^T$ such that $\mathbf{y}_{=k/2} \prec \mathbf{1} - \mathbf{b}_i$ **do**
7: **for** $1 \leq i \leq m$ **do**
8: Initialize an empty set $\mathfrak{Q}_i$.
9: For $1 \leq j \leq n$, if $r_i \equiv (j + \mathbf{c}_i \cdot \mathbf{s}_2) \bmod n$, update $\mathfrak{Q}_i \leftarrow \{j\} \cup \mathfrak{Q}_i$.
10: Insert each string $q \in \mathfrak{Q}_1 \times \cdots \times \mathfrak{Q}_m$ in the hash table along with $\mathbf{s}_2$.
11: For each collision in the hash table, construct $\mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2$ and update $\mathfrak{S} \leftarrow \{\mathbf{s}\} \cup \mathfrak{S}$.

12: Output $\mathfrak{S}$.

Once Algorithm 2 is executed, we need another algorithm to uniquely determine the vector in $\mathfrak{S}$ that satisfies $m \geq \log_d(n\binom{n+k-1}{k})$ challenge-response pairs.

Algorithm 3.
Input: The set $\mathfrak{S}$ and $m \geq \log_d(n\binom{n+k-1}{k})$ challenge-response pairs.
1: **for** each $\mathbf{s} \in \mathfrak{S}$ **do**
2: If $\mathbf{c}_i \cdot \mathbf{s} \equiv r_i \bmod n$ for $1 \leq i \leq m$, output $\mathbf{s}$ and halt.

The memory requirement of Algorithm 2 is $O(n\binom{n+k/2-1}{k/2})$. The combined running time of Algorithm 2 and 3 is:

$$O\Big(n\left(2mn\binom{n/2+k/2-1}{k/2} + \left(\frac{n}{d}\right)^m \binom{n/2+k/2-1}{k/2}\right) + \log_d\left(n\binom{n+k-1}{k}\right)\frac{\binom{n+k-1}{k}}{d^m}\Big)$$

Neglecting the logarithmic terms as well as the quadratic term in n, we get:

$$O\left(\frac{n^{m+1}}{d^m}\binom{n/2+k/2-1}{k/2} + \frac{\binom{n+k-1}{k}}{d^m}\right)$$

Following a similar procedure to that of [1, §6], in Appendix A, we show that to minimize this quantity the optimum value of m for Algorithm 2 is:

$$m = \frac{\ln\left(\frac{\binom{n+k-1}{k}\ln d}{\binom{n/2+k/2-1}{k/2}\ln(n/d)}\right)}{\ln n} - 1$$

And this value of m gives the running time:

$$O\left(\binom{n+k-1}{k}^{1-\ln d/\ln n}\binom{n/2+k/2-1}{k/2}^{\ln d/\ln n}\right)$$

In contrast, if we use the meet-in-the-middle algorithm from [1], we get a running time of:

$$O\left(\binom{n+k-1}{k}^{1-\ln d/\ln n}\binom{n+k/2-1}{k/2}^{\ln d/\ln n}\right)$$

which is considerably larger in the second term. Table 1 shows various choices of the parameters n and k and the resulting combined time and space complexity of Algorithms 2 and 3. We assume $d = 10$ and the time and space complexities are represented by the symbols τ and μ respectively.

Table 1. The time complexity τ and space complexity μ of the time-memory tradeoff attack with the optimum value of m against n and k.

n	k	τ	μ	n	k	τ	μ	n	k	τ	μ
100	10	2^{33}	2^{33}	120	10	2^{36}	2^{35}	150	10	2^{39}	2^{37}
	20	2^{55}	2^{52}		20	2^{60}	2^{55}		20	2^{65}	2^{58}
	30	2^{72}	2^{67}		30	2^{79}	2^{71}		30	2^{86}	2^{76}
200	10	2^{43}	2^{39}	300	10	2^{48}	2^{43}	500	10	2^{55}	2^{47}
	20	2^{72}	2^{63}		20	2^{83}	2^{69}		20	2^{96}	2^{77}
	30	2^{97}	2^{83}		30	2^{112}	2^{92}		30	2^{132}	2^{104}

5.4 Comparative Time Complexities

The main motivation behind our protocol was to increase Q relative to d in Equation 4 without compromising too much on usability. If Q roughly equals the square of d, then we can choose smaller values of k, as the time complexity of the attack will increase. This is not straightforwardly possible in the k-weight HB protocol and the sum of k mins protocol. For instance, if $Q = 100 = 10^2 = d^2$, these protocols will require k additions of 2 digit numbers in a single round. This is prohibitively difficult for most humans since the additions have to be performed mentally. Our protocol achieves this by shifting the computations to the locations rather than the values of those locations. At each step, the user only has to add a 2 or 3 digit number to a single digit number. As a result, usability is preserved while the time-memory tradeoff attacks perform worse in our case. Table 2 shows a direct comparison of the three protocols in terms of the time-complexity of our attack. The time complexity of the meet-in-the-middle attack from [1] is labeled "Old", whereas our attack is labeled "New". As can be seen, our attack is efficient than the previous attack by a few orders of magnitude. Time-memory tradeoff attacks is one way to attack our protocol. The next section looks at a different way to attack the protocol.

5.5 Significance of the Jump Constant a

Let $\hat{r}$ denote a location. Clearly it is an integer modulo n. We first attempt to find the probability distribution of obtaining $\hat{r}$ as a sum of the values of k locations, ignoring the starting location and hence the jump constant a. To this end, let $p(k, \hat{r})$ be the probability that $\hat{r}$ is the final location after the sum of the values of k locations as in our protocol. In other words, it denotes the probability that $\hat{r}$ is the sum of k integers (not necessarily unique): $s_1, \ldots, s_k \in \mathbb{Z}_d$. Clearly, $p(1, \hat{r}) = 1/d$ for $0 \leq \hat{r} \leq d-1$ and $p(1, \hat{r}) = 0$ for $d \leq \hat{r} \leq n-1$. For any subsequent k, we see that the probability $p(k, \hat{r})$ can be obtained by:

$$p\left(k, \hat{r}\right) = \sum\nolimits_{i=0}^{n-1} p\left(k-1, \hat{r} - i \bmod n\right) p\left(1, i\right)$$

Table 2. The time complexities of the time-memory tradeoff attacks on the k-weight HB protocol, the sum of k mins protocol and our protocol.

		k-weight HB		Sum of k mins		Our Protocol	
n	k	Old	New	Old	New	Old	New
100	8	2^{22}	2^{18}	2^{24}	2^{22}	2^{30}	2^{28}
	12	2^{30}	2^{24}	2^{34}	2^{31}	2^{41}	2^{38}
	16	2^{37}	2^{29}	2^{45}	2^{41}	2^{51}	2^{47}
200	8	2^{26}	2^{22}	2^{28}	2^{26}	2^{37}	2^{36}
	12	2^{36}	2^{30}	2^{40}	2^{37}	2^{52}	2^{49}
	16	2^{46}	2^{37}	2^{53}	2^{49}	2^{65}	2^{61}
300	8	2^{28}	2^{24}	2^{30}	2^{28}	2^{42}	2^{40}
	12	2^{40}	2^{34}	2^{44}	2^{41}	2^{58}	2^{56}
	16	2^{50}	2^{42}	2^{57}	2^{53}	2^{73}	2^{70}

This is similar to [1, §3.2]. The following dynamic programming algorithm of time complexity $O(kn^2)$ then computes these probabilities:

Algorithm 4.

Input: n,k and d.

1: Assign $p(1,\hat{r}) \leftarrow 1/d$ for $0 \le \hat{r} \le d-1$ and $p(1,\hat{r}) \leftarrow 0$ for $d \le \hat{r} \le n-1$.
2: **for** $2 \le i \le k$ **do**
3: **for** $1 \le j \le n$ **do**
4: $p \leftarrow 0$.
5: **for** $1 \le l \le n$ **do**
6: $p \leftarrow p + p(i-1, j-l \bmod n)p(1,l)$.
7: $p(i,j) \leftarrow p$.
8: Output $p(k,\hat{r})$ for $0 \le \hat{r} \le n-1$.

Now, let $q(a,k,\hat{r})$ denote the probability of obtaining the location $\hat{r}$ as the sum of k integers and the starting location, thus including the jump constant a. Then, we can see that:

$$q\left(a,k,\hat{r}\right) = \frac{1}{d}\sum\nolimits_{i=0}^{d-1} p\left(k,\hat{r}+ia \bmod n\right)$$

Let U denote the uniform distribution over $\mathbb{Z}_n$. Let Q denote the distribution of the $q\left(a,k,\hat{r}\right)$'s. $\Delta(Q,U) = \frac{1}{2}\sum_{i=0}^{n-1}\left|q\left(a,k,i\right) - \frac{1}{n}\right|$ is defined as the statistical distance between the two probability distributions.

Lemma 3. *Fix a k. Suppose d divides n. Then $\Delta(Q,U)$ is minimum if $a = \frac{n}{d}$.*

Proof. a divides n into d blocks, each of length a. Since the starting location is uniform over $\mathbb{Z}_d$, its product with a will then be uniformly distributed over n. If $a \neq \frac{n}{d}$, then either $ad < n$ or $ad > n$. In both cases, the numbers between ad and n will have different probabilities of occurence than the remaining numbers. □

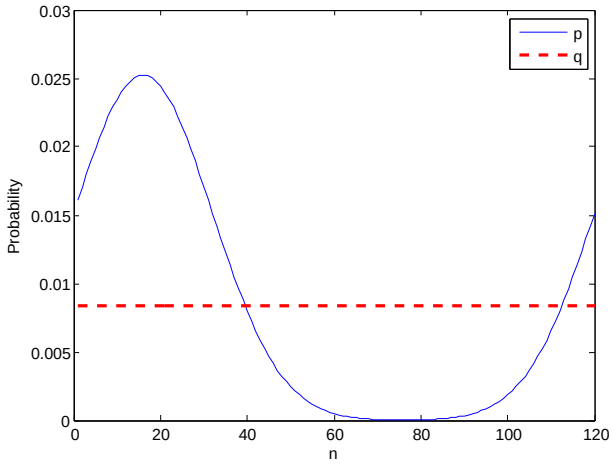

Fig. 2. The jump constant a makes the distribution nearly uniform over n.

Figure 2 shows the distribution of $q(a, k, \hat{r})$ against $p(a, k, \hat{r})$ with $n = 120$, $k = 30$ and $a = 12$. Table 3 shows $\Delta(Q, U)$ with different values of n and k (a is chosen such that $n = ab$). Based on these results, we see that if the statistical

Table 3. The statistical distance $\Delta(Q, U)$ against n and k. Greater value of n requires larger value of k to make $\Delta(Q, U)$ small.

n	k	$\Delta(Q,U)$	n	k	$\Delta(Q,U)$	n	k	$\Delta(Q,U)$
100	10	3.5×10^{-16}	120	10	4.9×10^{-8}	150	10	1.0×10^{-4}
	20	7.0×10^{-16}		20	3.3×10^{-15}		20	1.6×10^{-8}
	30	9.9×10^{-16}		30	1.0×10^{-15}		30	2.5×10^{-12}
200	10	7.2×10^{-3}	300	10	9.7×10^{-2}	500	10	3.3×10^{-1}
	20	8.2×10^{-5}		20	1.5×10^{-2}		20	1.7×10^{-1}
	30	9.3×10^{-7}		30	2.3×10^{-3}		30	8.8×10^{-2}

distance is small, an adversary can distinguish from the uniform distribution after observing $\frac{1}{\Delta(Q,U)}$ challenge-response pairs on average. The adversary can then (possibly) use some statistical methods to guess the starting location and then use the following algorithm to guess the answer to a challenge probabilistically:

Algorithm 5.

Input: n, k, d and a challenge $\mathbf{c}$.

1: Run Algorithm 4 to obtain an interval of locations, $\delta \subseteq [0, n-1]$, such that $\sum_{\hat{r} \in \delta} p\,(k, \hat{r}) = p\,(\delta)$, for some probability $p(\delta)$.

2: Guess a starting location and *shift* the interval δ accordingly.
3: Given a challenge $\mathbf{c}$, pick a location $\hat{r}$ uniformly at random from δ and output $\mathbf{c}[\hat{r}]$.

Then the success probability of this algorithm is:

$$\frac{p\,(\delta)}{n\delta} + \frac{(1 - p\,(\delta))}{n^2}$$

Considering $p(\delta)$ to be high, we can see that the quantity $1/n\delta$ is less than $1/n^2$. If the adversary knows the starting location, then the probability of success becomes $1/\delta$. As an example, for the parameters used in Figure 2, if $\delta = [0, 49] \cup [100, 119]$, then $p(\delta) = 0.93$ and the success probability of the algorithm is: 1.34×10^{-4} which is considerably better than the naive guess which succeeds with probability 7.0×10^{-5}.

Notice that each session in our protocol consists of m rounds. Therefore, in light of the discussion above, we mandate the use of our protocol for $\frac{1}{m\Delta(Q,U)}$ sessions only, before secret renewal.

6 Usability

To demonstrate comparable usability, we use similar parameters as used in the experiment in [1]. We use $n = 200$, $k = 15$ and $m = 6$. With these parameters, the time and space complexity of our time-memory tradeoff attack is proportional to 2^{61} and 2^{54} respectively. The statistical distance $\Delta(Q, U)$ is 4.9×10^{-4}, which means that the quantity $\frac{(\Delta(Q,U))^{-1}}{m} \approx 340$. Thus, with these parameters our scheme can be used securely for at least 340 authentication sessions.

With $n = 200$, $k = 15$ and $m = 7$, the experiment done for the k-weight HB protocol by the authors in [1] gave an average time of 166 seconds. Notice that there are $m = 7$ rounds instead of 6. This is important to add noise into the answer. The user sends the wrong answer to one of the challenges. To compare with our protocol, we can see that apart from the starting location, the user has to add two numbers for each secret location. One of these numbers is in the range $[0, 199]$ and the other is in the range $[0, 9]$. Thus arguably, adding a single digit number to a number in the range $[0, 199]$ will take approximately the same time, as we are well versed with doing such computations in our heads. For the starting location, we see that the user can move vertically (in a circular way) according to the digit corresponding to the starting location. The user reaches to a new location this way. The result of the remaining $k = 15$ locations can then be divided by 20 to get a quotient and a remainder. The quotient is the number of vertical steps and the remainder is the number of horizontal steps to be taken. The user can then follows these steps and output the digit corresponding to the location thus reached. Thus while this last step takes more time than the other steps, it can surely be done within half the time required for the computation of the $k = 15$ other secret locations. Now, one round of k-weight HB protocol takes

$166/7 \approx 23.7$ seconds on average. This implies that according to our argument, the computation of the last part takes ≈ 12 seconds. Thus, conjecturing that the calculations for the remaining $k = 15$ locations amounts to time 166, we can see that for $m = 6$ rounds, this amounts to a total time of ≈ 213 seconds.

From this discussion, we can say that for low values of α and β, our protocol is approximately $(\alpha, \beta, 213)$-human executable. While it takes slightly more time than the k-weight HB protocol, it is more usable as the user does not have to send a wrong answer with probability $1/7$, which is not possible for most humans. Furthermore, for these parameter choices, the time complexity of the time-memory tradeoff attack is proportional to 2^{37} in the case of k-weight HB protocol. In our case, the complexity is 2^{61}. The comparative time-complexity of the attack on the sum of k mins protocol with similar parameters is 2^{49}. Again, lower than the time-complexity for our protocol. To increase usability, one can further reduce m from 6 to 4 and get a time of approximately 143 seconds. Some of the common authentication mechanisms, such as PIN number authentication, use 4-digit numbers for security. We acknowledge the absence of actual experiments on users.

6.1 Handling Errors

Whenever humans are involved in performing computations, errors are unavoidable. With the default setting of the protocol, the legitimate user can be rejected if he/she does one mistake in any of the m rounds of the protocol. We can handle this by requiring that the user's answers be correct in most of the rounds. For instance, if $m = 6$, then the server accepts the user if 5 or more of the answers are correct. Although, the probability of success of a random guess attack will increase, it will still be considerably low. An interesting area of research would be to devise a protocol that handles user errors by using error correcting codes.

6.2 Suggested Parameters

Table 4 shows choices of parameter values for different security requirements and the resulting parameterized security against different attacks. In the table, m stands for the number of iterations (rounds) in one authentication session. R stands for the success probability of the random guess attack. B stands for the complexity of the brute force attack. τ/μ shows the time/space complexity of the time-space tradeoff algorithm. Finally, *Sessions*, represents the number of authentication sessions a particular secret can be used. It is obtained as $\frac{(\Delta(Q,U))^{-1}}{m}$. Notice that, space complexity can be a severe limitation as well ($2^{63}/8 \approx 10^{18}$, i.e. about 1 eta byte). For low and medium level security, the restriction on the number of sessions can be relaxed.

Notice that in comparison with some other protocols found in literature, the number of sessions is quite high. For instance, the cognitive authentication scheme of Weinshall [10] can only be used for approximately 40 sessions even when the size of the user's secret is as large as 150 [11, §4.1]. For more practical

Table 4. Suggested Parameters.

Security	n	k	m	R	B	τ/μ	$\Delta(Q,U)$	Sessions
Low	200	12	4	10^{-4}	2^{71}	$2^{49}/2^{44}$	2.9×10^{-3}	85
Medium	200	16	4	10^{-4}	2^{87}	$2^{61}/2^{54}$	4.9×10^{-4}	500
High	200	20	6	10^{-6}	2^{101}	$2^{72}/2^{63}$	8.2×10^{-5}	2,000
Paranoid	200	24	6	10^{-6}	2^{114}	$2^{83}/2^{71}$	1.4×10^{-5}	12,000

sizes of the secret, the number of allowable sessions is even lower. Similarly, Bai et al.'s scheme can be used for 10 authentication sessions only [12]. For these reasons, we have restricted our comparison to the two protocols in [1]. Li and Shum's protocols [14] seem promising, but their security has not been extensively analyzed; especially against time-memory tradeoff attacks.

7 Conclusion

Recently, many human identification protocols secure against passive adversaries have been proposed in the literature. However, they can only be used for a few authentication sessions before secret renewal. Furthermore, many of them lack a detailed security analysis. Some of them have ignored the impact of time-memory tradeoff attacks. We attempted to construct a protocol with security against time-memory tradeoff attacks in mind. The resulting protocol offers reasonable usability and good security. We acknowledge that the protocol can not be used frequently, as authentication seems to require about 2-3 minutes time. However, it can be used under certain circumstances such as when the user is using an insecure computer. An interesting question is whether improvements can be made to find a solution that achieves better security with progressively smaller values of parameters such as the size of the secret. Another area of interest is to find other variants of time-memory tradeoff attacks that can be applied to human identification protocols.

Acknowledgements. Hassan Jameel Asghar was supported by Macquarie University Research Excellence Scholarship (MQRES). Josef Pieprzyk was supported by the Australian Research Council under Grant DP0987734. The work of H. Wang is supported in part by the Australian Research Council under ARC Discovery Project DP0665035, the Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03 and the Singapore Ministry of Education under Research Grant T206B2204.

References

1. Hopper, N.J., Blum, M. Secure Human Identification Protocols. Advances in Cryptology - Asiacrypt 2001, Lecture Notes in Computer Science, Springer-Verlag, 52-66,

2001.

2. Hassan Jameel, Riaz Ahmed Shaikh, Heejo Lee and Sungyoung Lee: Human Identification Through Image Evaluation Using Secret Predicates. Topics in Cryptology - CT-RSA 07, Lecture Notes in Computer Science, Springer-Verlag. **4377** (2007) 67–84
3. Hassan Jameel, Riaz Shaikh, Le Hung, Yuan Wei, Syed Raazi, Ngo Canh, Sungyoung Lee, Heejo Lee, Yuseung Son, and Miguel Fernandes. Image-feature based human identification protocols on limited display devices. In Information Security Applications (WISA2008), volume 5379 of Lecture Notes in Computer Science, pages 211224. Springer, 2009.
4. Matsumoto, T., Imai, H.: Human Identification through Insecure Channel. Advances in Cryptology - EUROCRYPT 91, Lecture Notes in Computer Science, Springer-Verlag. **547** (1991) 409–421
5. Jermyn, I., Mayer, A., Monrose, F., Reiter, M., Rubin, A.: The design and analysis of graphical passwords. 8th USENIX Security Symposium (1999).
6. Wang, C.H., Hwang, T., Tsai, J.J.: On the Matsumoto and Imai's Human Identification Scheme. Advances in Cryptology - EUROCRYPT 95, Lecture Notes in Computer Science, Springer-Verlag. **921** (1995) 382–392
7. Matsumoto, T.: Human-computer cryptography: An attempt. 3rd ACM Conference on Computer and Communications Security, ACM Press. (1996) 68–75
8. Xiang-Yang Li, Shang-Hua Teng: Practical Human-Machine Identification over Insecure Channels. Journal of Combinatorial Optimization. **3** (1999) 347–361
9. Shujun Li, Heung-Yeung Shum: Secure Human-computer Identification against Peeping Attacks (SecHCI): A Survey. Unpublished report, available at Elsevier's Computer Science Preprint Server. (2002)
10. Daphna Weinshall: Cognitive Authentication Schemes Safe Against Spyware (Short Paper). 2006 IEEE Symposium on Security and Privacy. (2006) 295–300
11. Philippe Golle and David Wagner: Cryptanalysis of a Cognitive Authentication Scheme. Cryptology ePrint Archive, Report 2006/258. http://eprint.iacr.org/.
12. Xiaole Bai, Wenjun Gu, Sriram Chellappan, Xun Wang, Dong Xuan, Bin Ma: PAS: Predicate-Based Authentication Services Against Powerful Passive Adversaries. acsac,pp.433-442, 2008 Annual Computer Security Applications Conference, 2008.
13. Hirokazu Sasamoto, Nicolas Christin, and Eiji Hayashi. Undercover: Authentication Usable in Front of Prying Eyes. In Proceedings of the 2008 ACM Conference on Human Factors in Computing Systems (CHI 2008), pages 183-192. Florence, Italy, April 2008.
14. Shujun Li and Heung-Yeung Shum: Secure human-computer identification (interface) systems against peeping attacks:SecHCI. IACRs Cryptology ePrint Archive: Report 2005/268, August 2005.
15. D. Stinson. Some Baby-Step Giant-Step Algorithms for the Low Hamming Weight Discrete Logarithm Problem. Math. Comp., 71:379391, 2002.
16. G. Agnew, R. Mullin, I. Onyschuk, and S. Vanstone, An Implementation for a Fast Public-Key Cryptosystem, J. Cryptography, vol. 3, 1991.
17. A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. Handbook of applied cryptography, CRC Press, 1997.

A Optimum Value of m

We have:

$$\frac{n^{m+1}}{d^m}\binom{n/2+k/2-1}{k/2}+\frac{\binom{n+k-1}{k}}{d^m}$$

Let $C_1=\binom{n+k-1}{k}$ and $C_2=\binom{n/2+k/2-1}{k/2}$. Taking the first derivative test to find the minimum:

$$\begin{aligned}
&nC_2\frac{\mathbf{d}}{\mathbf{d}m}\left(\frac{n}{d}\right)^m+C_1\frac{\mathbf{d}}{\mathbf{d}m}\left(\frac{1}{d^m}\right)=0\\
&\Rightarrow nC_2\left(\frac{n}{d}\right)^m\ln\left(\frac{n}{d}\right)+C_1\left(\frac{1}{d^m}\right)\ln\left(\frac{1}{d}\right)=0\\
&\Rightarrow nC_2\left(\frac{n}{d}\right)^m\ln\left(\frac{n}{d}\right)=\frac{C_1}{d^m}\ln d\\
&\Rightarrow n^{m+1}=\frac{C_1\ln d}{C_2\ln\left(n/d\right)}\\
&\Rightarrow m=\frac{\ln\left(\frac{C_1\ln d}{C_2\ln(n/d)}\right)}{\ln n}-1\\
&\Rightarrow m=\frac{\ln\left(\frac{\binom{n+k-1}{k}\ln d}{\binom{n/2+k/2-1}{k/2}\ln(n/d)}\right)}{\ln n}-1 \qquad (5)
\end{aligned}$$

Let,

$$A=\frac{\binom{n+k-1}{k}\ln d}{\binom{n/2+k/2-1}{k/2}\ln\left(n/d\right)} \qquad (6)$$

Then putting the value of A from Equation 6 into Equation 5, we get:

$$m=\frac{\ln A}{\ln n}-1 \qquad (7)$$

Also re-arranging Equation 7 gives us:

$$n^{m+1}=A \qquad (8)$$

Now let,

$$\begin{aligned}
W&=\frac{n^{m+1}}{d^m}C_2+\frac{C_1}{d^m}\\
\Rightarrow d^mW&=n^{m+1}C_2+C_1 \qquad (9)
\end{aligned}$$

Putting the value of n^{m+1} from Equation 8 into Equation 9:

$$d^mW=AC_2+C_1$$

Now, putting the value of A from Equation 6, we have:

$$\begin{aligned} d^m W &= \frac{C_1 \ln d}{C_2 \ln (n/d)} C_2 + C_1 \\ \Rightarrow d^m W &= C_1 \left(\frac{\ln d}{\ln (n/d)} + 1 \right) \\ \Rightarrow d^m W &= C_1 \left(\frac{\ln d + \ln (n/d)}{\ln (n/d)} \right) \\ \Rightarrow d^m W &= C_1 \left(\frac{\ln n}{\ln (n/d)} \right) \end{aligned} \tag{10}$$

Let,

$$D = \frac{\ln n}{\ln (n/d)} \tag{11}$$

Then Equation 10 becomes:

$$\begin{aligned} d^m W &= C_1 D \\ \Rightarrow m \ln d + \ln W &= \ln C_1 + \ln D \\ \Rightarrow \ln W &= \ln C_1 + \ln D - m \ln d \end{aligned} \tag{12}$$

Putting in the value of m from Equation 7 into Equation 12:

$$\begin{aligned} \ln W &= \ln C_1 + \ln D - \left(\frac{\ln A}{\ln n} - 1 \right) \ln d \\ \Rightarrow \ln W &= \ln C_1 + \ln D - \frac{\ln A}{\ln n} \ln d + \ln d \\ \Rightarrow W &= C_1 D \left(e^{- \ln A} \right)^{\ln d / \ln n} d \\ \Rightarrow W &= C_1 D d A^{- \ln d / \ln n} \end{aligned} \tag{13}$$

Now, putting the values of D and A from Equations 11 and 6 respectively into Equation 13, we get:

$$\begin{aligned} W &= C_1 \cdot \frac{\ln n}{\ln (n/d)} \cdot d \cdot \left(\frac{C_1 \ln d}{C_2 \ln (n/d)} \right)^{- \ln d / \ln n} \\ \Rightarrow W &= \frac{\ln n}{\ln (n/d)} \left(\frac{\ln d}{\ln (n/d)} \right)^{- \ln d / \ln n} d C_1^{1 - \ln d / \ln n} C_2^{\ln d / \ln n} \end{aligned}$$

And by neglecting the logarithmic terms in the product:

$$W = O \left(d C_1^{1 - \ln d / \ln n} C_2^{\ln d / \ln n} \right)$$

Assuming d to be small, and putting in the values of C_1 and C_2, we see that the total computational time required is:

$$O \left(\binom{n + k - 1}{k}^{1 - \ln d / \ln n} \binom{n/2 + k/2 - 1}{k}^{\ln d / \ln n} \right) \tag{14}$$