Mitigating Dictionary Attacks on Password-Protected Local Storage

Ran Canetti Shai Halevi Michael Steiner

IBM T.J. Watson Research Center

August 16, 2006

Abstract

We address the issue of encrypting data in local storage using a key that is derived from the user's password. The typical solution in use today is to derive the key from the password using a cryptographic hash function. This solution provides relatively weak protection, since an attacker that gets hold of the encrypted data can mount an off-line dictionary attack on the user's password, thereby recovering the key and decrypting the stored data.

We propose an approach for limiting off-line dictionary attacks in this setting *without relying on secret storage or secure hardware.* In our proposal, the process of deriving a key from the password requires the user to solve a puzzle that is presumed to be solvable only by humans (e.g, a CAPTCHA). We describe a simple protocol using this approach: many different puzzles are stored on the disk, the user's password is used to specify which of them need to be solved, and the encryption key is derived from the password *and the solutions of the specified puzzles.* Completely specifying and analyzing this simple protocol, however, raises a host of modeling and technical issues, such as new properties of human-solvable puzzles and some seemingly hard combinatorial problems. Here we analyze this protocol in some interesting special cases.

Keywords. CAPTCHAs, Passwords, Puzzles, Storage encryption, Human-computer interaction.

1 Introduction

The motivation for this work is the common situation where we need to protect local storage that is physically readable by anyone, and moreover the only source of secret key material are the human users of the system. For example, think of a multi-user system where each user has an account, and where a browser lets users store personal information on the shared disk under the protection of a password. Another example is a laptop whose disk is searchable when captured and access to data is protected by a password. The common solution for this case is to derive a cryptographic key from the user-supplied password (possibly together with a public, locally stored salt), and use that key to encrypt the information (see e.g. [Kal00]).

This practice is quite problematic, however, since an attacker can perform dictionary searches for the correct password. In contrast to the case of password-based key exchange where off-line dictionary attacks can be effectively mitigated using cryptographic tools, here the lack of any secret storage seems to make such attacks inevitable. Thus, typical applications use a key-derivation-function such as SHA1 repeated a few thousand times to derive the key from the password, in the hope of slowing down off-line dictionary attacks. Quoting Wikipedia:

> *Key derivation functions are also used in applications to derive keys from secret passwords or passphrases, which typically do not have the desired properties to be used directly as cryptographic keys. In such applications, it is generally recommended that the key derivation function be made deliberately slow so as to frustrate brute-force attack or dictionary attack on the password or passphrase input value.*

`http://en.wikipedia.org/wiki/Key_derivation_function`

Although helpful, this approach is limited, as it entails an eternal cat-and-mouse chase where the number of iterations of SHA1 continuously increases to match the increasing computing powers of potential attackers. (This approach can be thought of as a naive instantiation of the "pricing via processing" paradigm of [DN92].)

This work aims to improve the security of local storage by relying on the human user for more than just supplying the initial password. Namely, we envision an interactive key-generation process involving a human user and the automated program, at the end of which a key is generated. Specifically, we propose to make use of puzzles that are easily solvable by humans but are hard to solve for computers, as in [Naor96, vAB+03]. That is, the user supplies a password, then it is asked to solve some puzzles, and the key is derived from both the password and the solutions of these puzzles.

One approach for combining puzzles and passwords was proposed by Pinkas and Sandler [PS02] in the context of a client-server interaction. In their solution the server asks the client to solve some puzzles, and if the client solves these puzzles correctly then the password is used in the usual way. This approach cannot be used in our setting, however, since it requires the server to keep the correct answers in storage for the purpose of comparing them with the solutions provided by the user. In our setting there is no server that can check the user's answers, and no secret storage to keep the solutions to the challenge puzzles.

Another potential solution, proposed by Stubblefield and Simon [SS04], is to use puzzles (called Inkblots) to which each individual has its own personal and repeatable answers that are unpredictable even by other humans. Then, a (hopefully) high entropy key can be derived from the user's solutions. In essence, this method uses the answers to the puzzles as a high-entropy password. If feasible, such an enhanced password generation mechanism is indeed very attractive, but the strong unpredictability requirements severely limit the class of potentially appropriate puzzles.

We combine the traditional password mechanism with human-solvable puzzles in a different way: At system setup, a large number of puzzles are generated and stored, *without their solutions.* The user-supplied password is then used to choose a small number of puzzles out of the stored ones. The user is asked to solve these puzzles, and the key is derived from the password *and the solutions to the puzzles.* The point here is to force the adversary to solve a considerable number of puzzles per each password guess, thus limiting its power to exhaustively search for the password.

The class of puzzles that are useful for our scheme is rather broad. We only need to be able to automatically generate puzzles, have individual human users answer these puzzles in a consistent way across time, and have the answer be unpredictable to a computer. In particular, there is no need to generate puzzles together with their answers, as required in the case of CAPTCHAs [vAB+03]. In fact, there need not even exist a single "correct" answer; each individual might have its own answer, as long as it is repeatable. On the other hand, there is no need that answers by one human are unpredictable by other humans, as required for Inkblots. We call this class of puzzles by the generic term human-only solvable puzzles (HOSPs). The Appendix suggests ideas for HOSPs that are neither good CAPTCHAs nor good Inkblots but might be suitable for our scheme.

As simple as this scheme sounds, analyzing it (or even completely specifying it) takes some work. For starters, one needs to determine how many puzzles to store on the disk and how many of these should the legitimate user solve in order to get access to the encrypted data. Also, one needs to specify the function that maps user passwords to sets of puzzles and the function that maps puzzle-solutions to cryptographic keys. (We call the first function Expand and the second function Extract. These suggestive names are motivated later.) Still more difficult questions are what hardness properties we need of the puzzles in use for this scheme to be secure, or even what attack model should be considered here. We elaborate on these issues below.

1.1 Formalizing the attack model

The overall goal of the scheme is to generate a pseudorandom key, meaning that feasible attackers should only have small advantage in distinguishing the key from random. The term "feasible" is typically defined as probabilistic polynomial time (PPT), so it is tempting to define security in the standard way:

> Security, first attempt: *A scheme is secure if given everything that is stored on the disk and a candidate key, no PPT algorithm can distinguish between the case where the key is generated by the scheme and the case where the key is chosen at random (except perhaps with small advantage).*

However, this definition ignores the fact that the attacker may have access to humans that can solve puzzles for it. Indeed, a known attack against systems that deploy CAPTCHAs is for the attackers to ship the same CAPTCHAs to their own web sites, asking their visitors to solve them. We thus need to extend the model, allowing the attacker to recruit some humans for help.

Devising a rigorous model that captures attackers with human help raises various issues, both technical and philosophical. Indeed, we do not even have good models for describing *honest* human participants in our protocols, let alone potentially malicious human attackers. For instance, should human help be restricted to solving puzzles? If so, how to model the quality of the answers? Are they always "correct", in the sense that one human can predict the answers of another human? Does it make sense to restrict the attacker to ask for solutions of puzzles that are actually used in the scheme, or can it queries for solutions to other puzzles as well? Is it legitimate to use a human in order to determine which puzzles to ask solutions for? The problem becomes even more intricate when one wishes to somehow *quantify* the amount of human help involved.

This work takes a rather simplistic approach, assuming that humans are only accessed as puzzle-solving oracles. This assumption can be justified to some extent if we view the attacker's human helpers not as malicious but rather as basically honest users that were tricked into helping the attack. Still, one should keep in mind that this model is not completely realistic. We further simplify the model by assuming that the oracle always returns the "correct" answer provided by the legitimate user. Furthermore, we only count the overall number of puzzles that the attacker asks to solve, not differentiating between "hard" and "easy" ones. That is, our notion of security has the following flavor:

> Security, better attempt: *A scheme is secure if given everything that is stored on the disk and a candidate key,* and given limited access to puzzle-solving humans, *no PPT algorithm can distinguish between the case where the key is generated by the scheme and the case where the key is chosen at random (except perhaps with small advantage).*

An additional simplifying assumption that we make is that the adversary only queries its oracle on puzzles that are explicitly used in the scheme (i.e., puzzles that are written on the disk). That is, we do not consider the possibility that an adversary may be able to modify a puzzle z to obtain a puzzle z', so that the solution of the original puzzle may be easier to find given the solution for z'. (Indeed, with existing CAPTCHAs one may slightly change the puzzle without changing the solution at all.) Following [DDN00], we refer to this concern as malleability of puzzles. Intuitively, a useful puzzle systems should be "non-malleable", in the sense that any query to the human oracle can help in solving at most a single puzzle out of the puzzles used in the system.

This seems to be the first time that such malleability issues are raised in the context of human-solvable puzzles (in fact, in the context of any non-interactive puzzle system), and that formalizing reasonable non-malleability properties for puzzles is an interesting challenge. This is a topic for future research.

Computational hardness of puzzles. The assumption that we make on the hardness of puzzles is that given a random puzzle z (and without any help from humans), it is hard to distinguish the real solution of z from a "random solution" taken from some distribution. (Clearly, this is stronger than just assuming that computing solutions is hard.)

In a little more details, a puzzle is essentially a sampleable distribution of problems with the additional property that humans can associate a solution with each problem in a repeatable way across time. This last property is captured in the model by having a puzzle-solving oracle H (for Human), which is an arbitrary deterministic function. The solution to a puzzle z is then defined as $H(z)$. The hardness assumption is formulated roughly by saying that given a random puzzle z, no PPT algorithm (that has no oracle access) can distinguish the "right solution" $H(z)$ from a "random solution" that is drawn from some distribution. The amount of hardness is measured in terms of the min-entropy of this distribution, denoted μ. See more discussion in Section 2.1.

A generic attack. We illustrate the capabilities of the attacker in our model by describing a generic attack against our scheme. Assume that there is an algorithm that given a puzzle generates a set of M "plausible solutions" that is guaranteed to include the right solution. (Hence this attack treats the puzzle system as if it has $\mu = \log M$ bits of pseudo-entropy.) Also, assume that the attacker has a dictionary D that is guaranteed to include the user's password.

The attacker gets an alleged key k^* and some n puzzles $z_1, \ldots, z_n$, and it needs to decide if the key is "real" or "random". It goes over the entire dictionary, expanding each password into the corresponding set of puzzles. Then it goes over all plausible solutions to each puzzle, extracts the key from each vector of plausible solutions, and keeps a list of all the solution-vectors that yield the input key k^*. These are the "consistent solutions".

The attacker then enlists the help of its human oracle in order to narrow down this list of consistent solutions. It adaptively queries the human oracle for solutions to puzzles, purging from the list all the vectors that contain a wrong solution to any puzzle. (The puzzles to be queried are chosen so as to maximize the information obtained by each query, e.g. by querying puzzle that appears in the most consistent solution vectors.) Finally, the attacker checks how many remaining consistent solutions are left for each password, and applies the maximum-likelihood decision rule to determine whether these numbers are more likely to be generated for a random key or for the "real key". The statistical advantage of this attack stems from the fact that for a "real" key k^* we expect to have one more consistent solution than for a "random" key (i.e., the actual solution).

It can be seen that to have significant advantage, the attacker must query its oracle on all but at most $|k^*|/\log M$ puzzles indexed by the actual password. Our analysis indicates that this is not just a property of this particular attack; we get essentially the same bound on the advantage of *any* attack.

1.2 Towards a fully specified scheme

Below we elaborate on how to determine the functions Expand (mapping passwords to puzzles) and $\mathsf{Extract}$ (mapping solutions and passwords to keys). This involves fixing a number of parameters, namely the number n of puzzles stored on the disk, the number ℓ of puzzles that a user is required to solve to obtain access to the data, and the length m of the generated key.

The Expand function. The resilience of the scheme against exhaustive password search comes from the need of the adversary to solve many puzzles (using its human oracle) in order to check each password guess. To this end, Expand must ensure that no large set of passwords is mapped to a small set of puzzles.

Let q be a bound on the number of puzzle queries that the attacker can make, and let μ be the assumed hardness of the puzzle system. It is clear that we must store on the disk $n \gg q$ puzzles, since we need to have many puzzles that the attacker "misses" (i.e., puzzles on which it does not query its human oracle). Let $I = \{i_1, i_2, \ldots, i_q\}$ be the indexes of puzzles that the attacker queries to its puzzles solver. We say that I "fully covers" a specific password pw if all the indexes in $\mathsf{Expand}(\mathsf{pw})$ are in the set I, and we say that I "almost covers" pw if at least $\ell - (m/\mu)$ of the indexes in $\mathsf{Expand}(\mathsf{pw})$ are in I. Our goal is to design the function Expand so that the attacker cannot have a small set of indexes that "almost covers" too many passwords.

This property is related to the quality of Expand as an expander graph: Consider a bipartite graph with passwords on the right and (indexes of) puzzles on the left, and where a password pw is connected to all the indexes in $\mathsf{Expand}(\mathsf{pw})$. Then a good expansion from right to left means that there is no large set of passwords that are fully covered by a small set of indexes (i.e., all their neighbors are included in a small set of indexes). The property that we need is stronger, namely that there is no large set of passwords that are even "almost covered" by a small set of indexes. It terms of expansion, we need that any subgraph of Expand (with the same set of nodes) in which the degree of nodes on the right is $\ell - \lceil m/\mu \rceil$ also has good expansion from right to left. This notion seems to be closely related to *fault-tolerant expanders* [DPPU88, AC88, U94]: the main difference is that fault-tolerant expanders need to remain "mostly expanding" when some vertices are removed, whereas we need the graph to remain an expander when some edges are removed.

In Section 3.2 we analyze the cover properties of a truly random Expand function (which essentially tells us what is the best that we can expect from any function) and also briefly discuss plausible explicit constructions for Expand.

The $\mathsf{Extract}$ function. Intuitively, the main property needed from function $\mathsf{Extract}$ is good randomness extraction, namely to compute a pseudorandom key from a set of solutions that is somewhat unpredictable to the attacker. When modeling hardness of puzzles by having the "right solutions" be indistinguishable from "random solutions" from a distribution with sufficient min-entropy, this intuition can be formalized in a straightforward way: namely as long as the input to $\mathsf{Extract}$ has sufficient min-entropy, the output should be random. Thus, it is sufficient if the function $\mathsf{Extract}$ is a (strong) randomness extractor [NZ96].

1.3 Security analysis

We provide two different analyses for the security of our construction. In each of these analyses we bound the advantage of an attacker as a function of the number of queries that it makes to its puzzle-solving oracle. (This is somewhat similar to the case of interactive password-based protocols where the advantage of an attacker is bounded as a function of the amount of interaction between the attacker and the honest participants.) In both analyses we reduce the security of the scheme to the assumption that an attacker cannot distinguish the "right solution" to random puzzles from "random solutions".

The first analysis (Section 4) uses a notion of security that follows the game-based approach for defining security of keys [BR93, BPR00]. The second analysis (Section 5) is done in the UC security model [Can01]. The "ideal functionality" that we present in this analysis follows the approach of [CHK$^+$05], in that it lets ideal-world adversary make only a limited amount of password queries. This analysis requires some additional properties from the puzzles and an additional "invertibility" property from $\mathsf{Extract}$; see details within.

Organization. Section 2 defines key-generation and recovery schemes (KGR) and human-only solvable puzzles. Section 3 presents the scheme and provides some combinatorial analysis of the properties of Expand. Sections 4 and 5 contain the two analyses of the scheme as described above, Section 6 has a concrete instantiation of the scheme, and Section 7 lists some open research problems. Appendix A suggests some potential implementations of human-solvable puzzles.

2 Password-Based Key Generation and Recovery Using Puzzles

Our goal is to generate a random encryption key that is recoverable by the legitimate user but still looks random to an attacker. Namely, we have two procedures: *key-generation* and *key recovery*, where the former can potentially store on the disk some information that the latter can use for recovery. In our setting, both of these procedures take as input the user's password (which should be thought of as a "weak secret"). Also, we allow them to interact with the legitimate user, modeled as a puzzle-solving oracle. The formal definition of the needed functionality proceeds as follows:

Definition 1 (Key Generation and Recovery) *A key generation and recovery scheme consists of two PPT algorithms with oracle access,* $(\mathsf{kGen}, \mathsf{kRec})$*:*

$(\mathsf{key}, S) \leftarrow \mathsf{kGen}^H(1^m, \mathsf{pw}, \mathsf{aux})$**.** *The randomized procedure* kGen *takes as input the security parameter and a password (and potentially also some other auxiliary input). It is given access to a human puzzle-solver and it outputs a key and (typically) some additional storage S.*

$\mathsf{key} \leftarrow \mathsf{kRec}^H(1^m, \mathsf{pw}, S)$**.** *The procedure* kRec *takes as input the security parameter, a password, and the additional storage S. It is given access to the oracle H and it returns the key.*

The pair $(\mathsf{kGen}, \mathsf{kRec})$ *is* correct *if for any oracle* H*, any* $k \in \mathbb{N}$, $\mathsf{pw} \in \{0,1\}^*$*, and every* (key, S) *in the support of* $\mathsf{kGen}^H(1^m, \mathsf{pw})$*, it holds that* $\mathsf{kRec}^H(1^m, \mathsf{pw}, S) = \mathsf{key}$*. When coupled with a specific function* H*, we sometime call the triple* $(\mathsf{kGen}, \mathsf{kRec}, H)$ *a Key-Generation-and-Recovery system (KGR for short).*

The key generation stage can be partitioned to a system set-up stage, which would include the generation of puzzles and is common to all users, and an actual generation stage where the key is generated from the password provided by the user. We do not formally enforce this partitioning to allow for potential optimizations that involve the user in the generation of the puzzles. We also note that the auxiliary input to the key-generation procedure is meant to allow the user to provide some personal input that might help generate (or "personalize") the puzzles; see discussion in the Appendix.

2.1 Human-Only Solvable Puzzles

For our purposes, a puzzle is a randomized puzzle-generation algorithm G with the additional property that random puzzles are solvable in a mostly consistent way by most humans. That is, the same person gives the same answer to a given puzzle at different times (although this consistent answer may vary from person to person). To enable asymptotic analysis, we let the puzzle generation algorithm G take as input the security parameter and also quantify the hardness of the resulting puzzles as a function of the security parameter.

Definition 2 (Puzzles) *A puzzle-system is a pair* (G, H)*, where* G *is a randomized puzzle generator that takes as input* 1^m *(with* m *the security parameter) and an auxiliary input* aux *and outputs a puzzle,* $z \leftarrow G(1^m, \mathsf{aux})$*, and* H *is a solution function. That is, the value* $a = H(z)$ *is the* solution *of the puzzle* z*.*

In order to be useful for our application, we need the puzzle system to be consistently-solvable by humans but not easily solvable by computers. The specific hardness assumption that we make is that efficient algorithms cannot distinguish the right solution $H(z)$ from "random". We call this hardness assumption the *solution indistinguishability* of the puzzle system, and quantify it by the amount of randomness in the "random" solution.

Definition 3 (Solution indistinguishability) *Let* $\mu = \mu(m)$ *be a function of the security parameter. A puzzle system* (G, H) *has* μ bits of pseudo-entropy *if for every auxilary input* aux *and every puzzle* z *in the support of the output of* $G(1^m, \mathsf{aux})$ *there is a distribution* $R_{\mathsf{aux}}(z)$ *with min-entropy at least* $\mu(m)$ *such that the correct solution is indistinguishable from a solution taken randomly from* $R_{\mathsf{aux}}(z)$*, even when given both correct and random solution to polynomially many other puzzles. That is, for evry auxiliary input* aux *and every polynomial* $n = n(m)$ *the following two ensembles are computationally indistinguishable:*

$$\left\{ (z_1, a_1, (z_2, a_2, a_2'), \ldots, (z_n, a_n, a_n')) \;:\; \begin{array}{l} z_1, \ldots, z_n \xleftarrow{\$} G(1^m, \mathsf{aux}),\ a_1 \leftarrow H(z), \\ a_i \leftarrow H(z_i),\ a_i' \xleftarrow{\$} R_{\mathsf{aux}}(z_i)\ (i = 2 \ldots n) \end{array} \right\}_m$$

and

$$\left\{ (z_1, a_1', (z_2, a_2, a_2'), \ldots, (z_n, a_n, a_n')) \;:\; \begin{array}{l} z_1, \ldots, z_n \xleftarrow{\$} G(1^m, \mathsf{aux}),\ a_1' \xleftarrow{\$} R(z), \\ a_i \leftarrow H(z_i),\ a_i' \xleftarrow{\$} R_{\mathsf{aux}}(z_i)\ (i = 2 \ldots n) \end{array} \right\}_m$$

We note that if the puzzle system is a CAPTCHA where the puzzles can be generated together with their solution, and if the distributions $R(z)$ are efficiently sampleable given z, then there is no need for the tuples (z_i, a_i, a'_i). Also, a simple hybrid argument shows that t-fold repetition of (G, H) multiplies the pseudo-entropy by t:

Observation 1 *Let μ, t be functions of the security parameter where t is polynomially bounded. Let (G, H) be a puzzles system, and let (G^t, H^t) be the t-fold repetition of it, where t puzzles are generated and solved independently. If (G, H) has μ bits of pseudo-entropy then (G^t, H^t) has $t\mu$ bits of pseudo-entropy.*

Proof (sketch): We show that an adversary A with distinguishing advantage ϵ against the t-fold repetition (G^t, H^t) can be transformed to an adversary A' with comparable running time that has advantage ϵ/t against the original system (G, H).

A' gets as input $(z_1, a_1, (z_2, a_2, a'_2), \ldots, (z_n, a_n, a'_n))$ and it needs to decide if a_1 is real or random. It chooses at random an index $j \in [1, t]$ and sets

$$\begin{array}{lll} \tilde{z}_i = z_{i+1}, & \tilde{a}_i = a_{i+1} & \text{for } i = 1, \ldots, j-1 \\ \tilde{z}_j = z_1, & \tilde{a}_j = a_1 & \\ \tilde{z}_i = z_i, & \tilde{a}_i = a'_i & \text{for } i = j+1, \ldots, t \end{array}$$

Then A' activates the t-fold distinguihser $A((\tilde{z}_1, \tilde{a}_1), \ldots, (\tilde{z}_t, \tilde{a}_t), (z_{t+1}, a_{t+1}, a'_{t+1}), \ldots, (z_n, a_n, a'_n))$ and outputs whatever A outputs. It is clear that the input to A when a_1 is real and the index is j is distributed the same as when a_1 is random and the index is $j + 1$, and the proof follows. □

Finally, we reiterate that the security models that are described in subsequent sections give the adversary access to the same oracle H that is used by the scheme. This represents the fact that, for analyzing security, we make the worst-case assumption that all humans provide the same answer to the same question, thus the adversary can employ other humans to reproduce the answers provided by the legitimate user. One can possibly model the case where people predict the answers of each other only in a "partial", or "noisy" way. Still, it is stressed that for the correctness of the scheme we only need that the same user answers consistently with itself.

3 The Scheme

The scheme uses a puzzle system (G, H) as above, where G is used directly by the key-generation procedure and H is assumed to be available as an oracle at both key-generation and key-recovery time.

The scheme depends on two internal parameters: the number of puzzles that are stored on the disk (denoted n) and the number of puzzles that the user needs to solve (denoted ℓ). We think of ℓ as a constant or a slowly increasing function, and $n = poly(m)$. In addition, we have a universe W of passwords (e.g., $W = \{0, 1\}^{160}$ if using SHA1 to hash the real password before using it.) W should not be confused with the potential dictionary $D \subset W$ from which passwords are actually chosen, which is not known when the scheme is designed. The scheme uses two randomized functions, $\mathsf{Expand}_{r_1} : W \to [n]^\ell$ and $\mathsf{Extract}_{r_2} : \{0, 1\}^* \to \{0, 1\}^m$, that are discussed later in this section. Given all these components, denote $n = n(m)$ and $\ell = \ell(m)$, and the scheme works as follows:

$\underline{\mathsf{kGen}^H(1^m, \mathsf{pw}, \mathsf{aux})}$**.** Generate n puzzles, $z_i \leftarrow G(1^m, \mathsf{aux})$, $i = 1, 2, \ldots, n$, and choose the seeds r_1, r_2 for Expand and $\mathsf{Extract}$ at random. Then set $\langle i_1, \ldots i_\ell \rangle \leftarrow \mathsf{Expand}_{r_1}(\mathsf{pw})$ and query H for the solutions, setting $a_{i_j} \leftarrow H(z_{i_j})$ for $j = 1, \ldots, \ell$. Finally, compute $\mathsf{key} \leftarrow \mathsf{Extract}_{r_2}(a_{i_1}, \ldots, a_{i_\ell}, \mathsf{pw})$

and then all the solutions are discarded. The "additional storage" that is saved to the disk consists of the puzzles $\langle z_1, z_2, \ldots, z_n \rangle$ and the seeds r_1, r_2.

$\underline{\mathsf{kRec}(1^m, \mathsf{pw}, \langle z_1, \ldots, z_n \rangle, r_1, r_2)}$. Compute the indexes $\langle i_1, ... i_\ell \rangle \leftarrow \mathsf{Expand}_{r_1}(\mathsf{pw})$, query the oracle H for the solutions $a_{i_j} \leftarrow H(z_{i_j})$ for $j = 1, \ldots, \ell$, and recover the key as $\mathsf{key} \leftarrow \mathsf{Extract}_{r_2}(a_{i_1}, \ldots, a_{i_\ell}, \mathsf{pw})$.

We remark that if the puzzle system in use is in fact a CAPTCHA system (where puzzles are generated together with their solution) then the key-generation procedure need not query the oracle H. Also, if the puzzle system is such that puzzles remain hard to solve even when the randomness of G is known (and Expand is modeled as a random oracle) then the puzzles need not be stored at all. Instead, the value $\mathsf{Expand}(\mathsf{pw})$ can be used as the randomness to G, thus generating the puzzles "on the fly".

3.1 The function Extract

The role of Extract is to extract an m-bit pseudorandom key from the pseudo-entropy in the human solutions to the puzzles. Given our solution-indistinguishability assumption (cf. Definition 3), this can be achieved by using a strong randomness extractor [NZ96] for the function Extract, as long as we are willing to live with some loss of pseudo-entropy (since to get a close-to-random m-bits output from an extractor you need $m' > m$ bits of min-entropy in the input). Below and throughout the analyses, we therefore assume that Extract is a strong extractor (e.g., a pairwise-independent hash function) and denote by m' the amount of min-entropy that is needed in the input to Extract in order for the output to be close to a uniform m-bit string.

3.2 The function Expand

The role of the function Expand is to map passwords to indexes of puzzles in such a way that the attacker would have to solve many puzzles (i.e., invoke the puzzle-solving oracle H many times) to check each new password guess. Specifically, our goal is to make sure that as long as the attacker does not make too many queries to its puzzle-solving oracle, most passwords would have enough unsolved puzzles to get $m' > m$ bits of pseudo-entropy. (Recall that m' is the amount of min-entropy that is needed in the input to Extract in order for the output to be close to a uniform m-bit string.)

To make this more precise, fix the randomness r_1 and think of the function $E = \mathsf{Expand}_{r_1}$ as a bipartite graph with the password universe W on the right and the indexes $\{1, ..., n\}$ on the left, and where each $\mathsf{pw} \in W$ is connected to all the indexes in $E(\mathsf{pw})$. If the puzzle system has μ bits of pseudo-entropy then we denote by $\ell^* \stackrel{\text{def}}{=} \lceil m'/\mu \rceil$ the number of puzzles that the attacker should miss in order for the key to be pseudo-random.

We say that a set I of indexes on the left almost covers a password $\mathsf{pw} \in W$ on the right if pw has no more than ℓ^* neighbors that are not in I. (I.e., pw has at least $\ell - \ell^*$ neighbors in I.) The *almost-cover* of a set I relative to mapping E, dictionary D and parameter ℓ^* is:

$$\mathsf{acvrSet}_{\ell^*}(I, E, D) \stackrel{\text{def}}{=} \{\mathsf{pw} \in D : \ |E(\mathsf{pw}) \setminus I| \leq \ell^*\}.$$

What we want is that for any set of $q \ll n$ puzzles $z_{i_1}, ..., z_{i_q}$ that the attacker has solutions for, and any (large enough) potential dictionary $D \subset W$, the set $I = \{i_1, ..., i_q\}$ almost covers only a small fraction of the passwords in D. We denote by $\mathsf{acvr}_{\ell^*}(q, E, D)$ the cover number of E, namely the fraction of passwords in D that can be almost-covered by q puzzles:

$$\mathsf{acvr}_{\ell^*}(q,E,D) \stackrel{\text{def}}{=} \frac{\max_{|I|=q} \left|\mathsf{acvrSet}_{\ell^*}(I,E,D)\right|}{|D|}$$

We would like the expected cover-number of Expand_{r_1} (over the choice of r_1) to be sufficiently small. The property of having a small cover number is related to the expansion of the graph E (from right to left): We want any large enough subset of the nodes on the right to have more than q neighbors on the left, even when removing ℓ^* edges from every node. In other words, every subgraph of E where the degree of nodes on the right is $\ell - \ell^*$ should be a good expander.

We again recall that this notion is related to fault-tolerant expanders [DPPU88, AC88, U94], except that we are interested in omitted edges rather than omitted vertices. In situations where the password universe is very small (e.g., when the passwords are PIN numbers), perhaps it is even possible to use deterministic fault-tolerant expanders. However, a simple counting argument shows that when the password universe is large then no deterministic construction is possible.

To get a sense of the obtainable parameters when the password universe is large, we first provide an analysis in the random-oracle model (i.e., assuming that Expand is a random function). Later we discuss solutions based on limited independence and speculate about other plausible constructions.

The cover number of a random function. Fix the parameters q, n, ℓ, ℓ^* and the dictionary D, and assume that E is a random function, mapping each $\mathsf{pw} \in D$ on the right to ℓ random indexes in $[n]$ on the left. For a fixed subset of indexes $I \subset [n]$, $|I| = q$, and a fixed password $\mathsf{pw} \in D$ consider the indicator variable

$$\chi_{I,\mathsf{pw}} = \begin{cases} 1 & \text{if } I \text{ almost covers } \mathsf{pw} \\ 0 & \text{otherwise} \end{cases}$$

so $|\mathsf{acvrSet}_{\ell^*}(I,E,D)| = \sum_{p \in D} \chi_{I,\mathsf{pw}}$.

Lemma 2 *Assume that $q < n$ and $\ell^* = (1-\alpha)\ell$ for some $\alpha > 0$. Then for every $\mathsf{pw} \in D$ and every $I \subset [n]$ with $|I| = q$ it holds that $\tau \stackrel{\text{def}}{=} E[\chi_{I,\mathsf{pw}}] \le \binom{\ell}{\alpha\ell} \cdot \left(\frac{q}{n}\right)^{\alpha\ell}$.*

Proof: In order for all but ℓ^* of the neighbors of pw to be in I we must have some set of $\ell - \ell^* = \alpha\ell$ edges of pw leading to I. For any specific set of $\alpha\ell$ edges this happens with probability $(q/n)^{\alpha\ell}$. Using the union bound, the probability that there exists such set of edges is at most $\binom{\ell}{\alpha\ell} \cdot \left(\frac{q}{n}\right)^{\alpha\ell}$. □

The following technical lemma bounds the probability of having a very large cover number in terms of various parameters of the system. This lemma depends on many parameters so as to make it applicable in many different settings. (In Corollary 4 below we give a somewhat simpler formula, and later we show an example of some specific setting.)

Lemma 3 *Fix $\ell, \ell^*, n, q \in \mathbb{N}$ such that $q < n$ and $\ell^* = (1-\alpha)\ell$ for some $\alpha > 0$, and denote $\epsilon \stackrel{\text{def}}{=} \binom{\ell}{\alpha\ell}\left(\frac{q}{n}\right)^{\alpha\ell}$. Also fix some finite set $D \subset W$. Then for every $\beta \in (0,1)$, it holds that*

$$\Pr_E\left[\mathsf{acvr}_{\ell^*}(q,E,D) > \beta\right] < \binom{n}{q} \cdot \left(\left(\frac{\epsilon}{\beta}\right)^{\beta} \left(\frac{1-\epsilon}{1-\beta}\right)^{1-\beta}\right)^{|D|}$$

where the probability is taken over choosing a random function $E : D \to [n]^{\ell}$.

Proof: Recall that for every fixed I (with $|I| = q$), the $\chi_{I,\mathsf{pw}}$'s that were defined above are independent zero-one variables with $E[\chi_{I,\mathsf{pw}}] \le \epsilon$. The Chernoff bound asserts that for this fixed I it holds for all $\beta \in (0,1)$

$$\Pr_E\left[\frac{|\mathsf{acvrSet}_{\ell^*}(I,E,D)|}{|D|} > \beta\right] < \left(\left(\frac{\epsilon}{\beta}\right)^\beta \left(\frac{1-\epsilon}{1-\beta}\right)^{1-\beta}\right)^{|D|}$$

Using the union bound, the probability that there exists a subset of indexes $I \subset [n]$ with $|I| = q$ and $|\mathsf{acvrSet}_{\ell^*}(I,E,D)|/|D| > \beta$ is at most $\binom{n}{q}$ times the expression above. □

Corollary 4 *Fix $\ell, \ell^* \in \mathbb{N}$ such that $\ell^* = (1-\alpha)\ell$, and also fix the ratios q/n and $|D|/n$, and denote $\epsilon \stackrel{\text{def}}{=} \binom{\ell}{\alpha\ell}\left(\frac{q}{n}\right)^{\alpha\ell}$. Let $\beta \in (0,1)$ be such that*

$$2^{H_2(q/n)}\left((\epsilon/\beta)^\beta(1+\beta)\right)^{|D|/n} < 1$$

where H_2 is the binary entropy function. Then $\Pr_E[\mathsf{acvr}_{\ell^*}(q,E,D) > \beta] < 2^{-O(n)}$.

Proof: Substituting in the expression from Lemma 3 the bounds $\binom{n}{q} < 2^{nH_2(q/n)}$ and

$$\left(\frac{1-\epsilon}{1-\beta}\right)^{1-\beta} < \left(\frac{1}{1-\beta}\right)^{1-\beta} < 1+\beta$$

we get

$$\begin{aligned}\Pr_E[\mathsf{acvr}_{\ell^*}(q,E,D) > \beta] &< 2^{nH_2(q/n)}\left(\left(\frac{\epsilon}{\beta}\right)^\beta(1+\beta)\right)^{|D|} \\ &= \left(2^{H_2(q/n)}\left(\left(\frac{\epsilon}{\beta}\right)^\beta(1+\beta)\right)^{|D|/n}\right)^n\end{aligned}$$

and since the expression in the last parentheses is a constant smaller than one, the corollary follows. □

An example. Assume that we have $|D| = n$, $q/n = 0.01$, $\ell = 8$ and $\ell^* = 4$. In this case $\epsilon = \binom{4}{8}(q/n)^4 = 7\cdot 10^{-7}$, and also $H_2(q/n) < 0.0808$. Substituting $\beta = 0.007$ in the expression from Corollary 4 we get

$$\Pr_E[\mathsf{acvr}_{\ell^*}(q,E,D) > 0.007] < \left(2^{0.0808}\cdot\left(\frac{7\cdot 10^{-7}}{0.007}\right)^{0.007}\cdot 1.007\right)^n \approx 0.9985^n \approx 2^{-n/462}$$

Assuming large enough value for n (e.g., $n > 5000$), the expected value of $\mathsf{acvr}_{\ell^*}(q,E,D)$ over the choise of E is no more than about 0.007.

Note that the trivial way of "almost covering" passwords in this case will be for the attacker to make four queries for each password, and since we assume that there are n passwords and we have $q/n = 0.01$ then the fraction of passwords that will be almost covered this way will be 0.01/4=0.0025. Hence, in this case the upper bound tells us that the attacker cannot do better than three times the obvious attack.

Limited independence. Storing a completely random function from the password universe W to the set of puzzle indexes $[n]$ is not realistic in most cases. A first attempt at obtaining a concrete construction is to replace a completely random mapping with an X-wise independent mapping for some X. (In terms of storage on the disk, it is acceptable to store a description of an $O(n)$-wise independent function, since we anyway need $O(n)$ storage to store the n puzzles.)

When Expand is ℓt-wise independent then the $\chi_{I,\mathsf{pw}}$'s from Lemma 2 are t-wise independent and one can use the t-moment inequality instead of Chernoff bound. The bound that we get for any $\delta > 0$ is

$$\Pr_E [\mathsf{acvr}_{\ell^*}(q, E, D) > \epsilon + \delta] \quad < \quad \binom{n}{q} \cdot \frac{\epsilon}{\delta^t |D|^{t/2}},$$

where again we have $\epsilon \stackrel{\text{def}}{=} 2^{H_2(\alpha)}(q/n)^{\alpha}$. Using an ℓn-wise independent mapping (so $t = n$) and setting $\delta = 2/\sqrt{|D|}$ we get

$$\Pr_E \left[\mathsf{acvr}_{\ell^*}(q, E, D) > \epsilon + \frac{2}{\sqrt{|D|}}\right] < \binom{n}{q} \cdot \frac{\epsilon}{2^n} < \; 2^{q \log n - n}$$

We thus get:

Lemma 5 *Fix $\ell, \ell^*, n, q \in \mathbb{N}$ such that $\ell^* = (1-\alpha)\ell$ for some $\alpha > 0$ and $q < n/(4 \log n)$, and fix a finite set D such that $|D| > n$. The expected value of $\mathsf{acvr}_{\ell^*}(q, E, D)$ over the choice of E as an ℓn-wise independent mapping from W to $[n]$ is at most $\epsilon + 2/\sqrt{|D|} + 2^{q \log n - n}$ where $\epsilon \stackrel{\text{def}}{=} \binom{\ell}{\alpha\ell}(q/n)^{\alpha\ell}$.*
$\Box$

More efficient constructions. Although feasible, the solution of using ℓn-wise independent mapping is far from being satisfactory, as it entails very large storage and computational cost. Providing more efficient constructions that are provably good is an open problem.

One possible direction here is to extend for our purposes the result of Alon et al. [ADM$^+$99]. In that work they considered a mapping from n balls to n buckets, and analyzed the size of the largest bucket. They proved that although "generic pairwise independent function" cannot ensure anything smaller than $n^{1/2}$, using a random linear mapping (over the binary field) has expected largest bucket of only $\tilde{O}(\log n)$. This can be thought of as a very special case of our application with $\ell = q = 1$, $\ell^* = 0$ and $|D| = n$. We thus speculate that perhaps using ℓ independent linear maps could give us a reasonable bound also for our application.

4 Game-Based Security Analysis

In this section we analyze the security of our scheme with respect to a "game-based" notion of security (as in [BR93]). In the formulation below the key generation scheme is run on a password pw that was randomly chosen from a dictionary. The adversary is then given the generated storage, plus a value that is either the real generated key or a random value of the same length. In addition, the adversary is given oracle access to H. The adversary's advantage in distinguishing between the two cases is measured as a function of the number of H-queries (q), and the size of the dictionary from which the passwords are chosen (d):

Definition 4 (Game-based security definition) *Let $\alpha : \mathbb{N} \times \mathbb{N} \to [0, 1]$ be a function, let $(\mathsf{kGen}, \mathsf{kRec}, H)$ be a key-generation and recovery system, and let $\mathcal{C}$ be a class of attackers with oracle access.*

We say that $(\mathsf{kGen}, \mathsf{kRec}, H)$ *is secure up to* α *with respect to* $\mathcal{C}$ *if for any attacker* $A \in \mathcal{C}$, *any polynomially related* $d, m, q \in \mathbb{N}$ *and any* $D \subset W, |D| = d$, *the following two probabilities differ by at most* $\alpha(d, q) + \mathsf{negl}(m)$, *where* negl *is a negligible function:*

$$\begin{aligned} p_{\mathsf{real}}(A) &\stackrel{\text{def}}{=} \Pr[\mathsf{pw} \in_R D,\ (\mathsf{key}, S) \leftarrow \mathsf{kGen}^H(1^m, \mathsf{pw}) \quad : \quad A^{H_q}(\mathsf{key}, S) = 1] \\ p_{\mathsf{rand}}(A) &\stackrel{\text{def}}{=} \Pr[\mathsf{pw} \in_R D,\ (\mathsf{key}, S) \leftarrow \mathsf{kGen}^H(1^m, \mathsf{pw}), \\ &\qquad\qquad \mathsf{key}' \in_R \{0,1\}^{|\mathsf{key}|} \quad : \quad A^{H_q}(\mathsf{key}', S) = 1] \end{aligned}$$

where H_q *answers the first* q *queries as* H *does, and answers later queries by* $\perp$.

Conservative adversaries. For our analysis to go through, we need to restrict the attacker to only query its puzzle-solving oracle H on puzzles that explicitly appear in the storage S of the scheme. This is done to prohibit "puzzle mauling attacks" as discussed in Section 2.1. Essentially, this restriction reflects the assumption that each H-query helps in answering only a single puzzle, and that getting an H-answer to a puzzle does not help solving another puzzle.

Definition 5 (Conservative adversaries) *An adversary against our KGR scheme is called* conservative *if it only queries its oracle on puzzles that are explicitly included in the "additional storage" output of the key-generation routine.*

Security statement. We are now ready to state our main result concerning the security of our scheme. Let m be the security parameter and let $\delta, \mu, d, \ell, m', n, q$ be other parameters (that may be functions of the security parameter), and denote $\ell^* \stackrel{\text{def}}{=} \lceil m'/\mu \rceil$. Below we assume that that $\binom{\ell}{\ell^*}$ is polynomial in m. (Recall that ℓ is the number of puzzles that the honest user needs to solve, so we typically think of it as a constant or $\log m$, hence assuming that $\binom{\ell}{\ell^*}$ is polynomial in m is reasonable.)

Let Expand_{r_1} be a randomized mapping of passwords to indexes and let $\mathsf{Extract}_{r_2}$ be a strong randomness extractor [NZ96], extracting m bits that are δ away from uniform given any distribution with m' bits of min-entropy. Lemma 6 below essentially asserts that as long as there are at least m' bits of pseudo-entropy in the puzzles that are mapped to the right password but are not queried by the attacker, the attacker cannot have any significant advantage in distinguishing real from random. Specifically, we show that the advantage of the attacker is essentially the fraction of passwords in the dictionary that are almost covered by its queries to the puzzle-solving oracle.

Lemma 6 *Under the conditions from above, the scheme from Section 3 using* Expand *and* $\mathsf{Extract}$ *is secure up to* α *with respect to the class of conservative adversaries, where* α *is defined as*

$$\alpha(d, q) \stackrel{\text{def}}{=} \max_{|D|=d} E_{r_1}\left[\mathsf{acvr}_{\ell^*}(q, \mathsf{Expand}_{r_1}, D)\right] + \delta \binom{\ell}{\ell^*}.$$

Proof: Let A be a conservative adversary, fix q, d and a size-d dictionary $D \subseteq W$, and recall that we denote

$$\begin{aligned} p_{\mathsf{real}}(A) &\stackrel{\text{def}}{=} \Pr[\mathsf{pw} \in_R D,\ (\mathsf{key}, S) \leftarrow \mathsf{kGen}^H(1^m, \mathsf{pw}) \quad : \quad A^{H_q}(\mathsf{key}, S) = 1] \\ p_{\mathsf{rand}}(A) &\stackrel{\text{def}}{=} \Pr[\mathsf{pw} \in_R D,\ (\mathsf{key}, S) \leftarrow \mathsf{kGen}^H(1^m, \mathsf{pw}), \\ &\qquad\qquad \mathsf{key}' \in_R \{0,1\}^{|\mathsf{key}|} \quad : \quad A^{H_q}(\mathsf{key}', S) = 1] \end{aligned}$$

where H_q answers the first q queries as H does, and answers later queries by $\perp$. Below we assume that $p_{\mathsf{real}}(A) > p_{\mathsf{rand}}(A)$ and denote $\alpha(A) \stackrel{\text{def}}{=} p_{\mathsf{real}}(A) - p_{\mathsf{rand}}(A)$. (If $p_{\mathsf{real}}(A) < p_{\mathsf{rand}}(A)$ then consider a different attacker where the outputs are reversed.)

Before we describe the reduction, it is convenient to consider a mental experiment with a modified attack process A' that is not implementable in our attack model but nonetheless has almost the same advantage as our "real attacker" A. Specifically, A' gets all the inputs that A gets *and in addition also the password* pw. It runs A with these inputs (sans pw) and when A halts with output b' then A' does the following:

1. If A queried its H-oracle on more than $\ell - \ell^*$ puzzles that are indexed by $\mathsf{Expand}_{r_1}(\mathsf{pw})$ then A' queries the oracle on all the remaining un-queried puzzles that are indexed by $\mathsf{Expand}_{r_1}(\mathsf{pw})$ (if any) and then outputs the bit 1.

2. Otherwise, A' queries its H oracle on *all but* ℓ^* of the un-queried puzzles that are indexed by $\mathsf{Expand}_{r_1}(\mathsf{pw})$, and then it outputs the same bit b'.

Observe that A' always queries its H oracle on either all the puzzles that are indexed by $\mathsf{Expand}_{r_1}(\mathsf{pw})$ or on all of them but ℓ^*.

Observe also that in both the real and the random games the probability that A' outputs 1 is no less than the probability that A outputs 1. Also in the random game the value of the password pw is independent of everything that A sees, and therefore the probability that pw is almost covered by the index set I of the queries that A makes is at most $\mathsf{acvr}_{\ell^*}(q, \mathsf{Expand}_{r_1}, D)$. Therefore we have $p_{\mathsf{rand}}(A') \leq p_{\mathsf{rand}}(A) + \mathsf{acvr}_{\ell^*}(q, \mathsf{Expand}_{r_1}, D)$ and $p_{\mathsf{real}}(A') \geq p_{\mathsf{real}}(A)$, and so

$$p_{\mathsf{real}}(A') - p_{\mathsf{rand}}(A') \geq \alpha(A) - \mathsf{acvr}(q, \mathsf{Expand}_{r_1}, D)$$

We show a reduction algorithm B that uses A' as a subroutine and has advantage at least

$$\beta = \frac{p_{\mathsf{real}}(A') - p_{\mathsf{rand}}(A')}{\binom{\ell}{\ell^*}} - \delta$$

in the solution-indistinguishability game against an ℓ^*-fold repetition of (G, H). By Observation 1, this implies the existence of an adversary with advantage β/ℓ^* against (G, H).

The reduction B gets as input $\langle (z_1, a_1), \ldots, (z_{\ell^*}, a_{\ell^*}), (z_{\ell^*+1}, a_{\ell^*+1}, a'_{\ell^*+1}) \ldots, (z_n, a_n, a'_n) \rangle$ and it needs to decide if the a_i's for $i = 1 \ldots \ell^*$ are all real or all random solutions. First B chooses at random a password $\mathsf{pw} \in_R D$ and random seeds r_1, r_2 for Expand, $\mathsf{Extract}$, respectively, and computes $\mathbf{i} = \langle i_1, \ldots, i_\ell \rangle \leftarrow \mathsf{Expand}_{r_1}(\mathsf{pw})$. Then B randomly permutes the indexes, subject to the condition that the indexs $1, \ldots, \ell^*$ (for which it knows only one solution) are mapped to indexes in $\mathbf{i}$.

Let $\vec{z}$ be the permuted vector of puzzles and let $a_{i_1}, \ldots, a_{i_\ell}$ be the solutions to the puzzles in $\vec{z}$ that are indexed by $\mathbf{i} = \mathsf{Expand}_{r_1}(\mathsf{pw})$. (Each a_{i_j} is either the only solution that B knows for the corresponding puzzle, or the "real solution" if B knows both solutions.) Then B computes $\mathsf{key} \leftarrow \mathsf{Extract}_{r_2}(a_{i_1}, \ldots, a_{i_\ell}, \mathsf{pw})$ and activates the modified attacker $A'(\mathsf{key}, r_1, r_2, \vec{z}, \mathsf{pw})$.

Whenever A' makes an oracle query to $H(z)$ on a puzzle for which B knows has both a real and a random solution, then B answers with the real solution that it knows. If at any point A' queries its H-oracle for a solution on a puzzle z for which B knows only one solution then B aborts the computation, outputting the bit 1. If B does not abort and A' halts with some output bit then B outputs the same bit.

When analyzing the advantage of B, it is more convenient to analyze another mental experiment where the choice of the indexes on which B aborts is delayed until the end. Consider B' that gets all the real and random solutions to all the n puzzles, and in addion a bit σ telling it if it needs to emulate B using "real" or "random" solutions. Namely, the input of B' is $\langle \sigma, (z_1, a_1, a'_1), \ldots, (z_n, a_n, a'_n) \rangle$ (and we denote by $\vec{z}$ the vector of puzzles). B' chooses at random a password $\mathsf{pw} \in_R D$ and random seeds r_1, r_2 for Expand, $\mathsf{Extract}$, respectively, and computes $\mathbf{i} = \langle i_1, \ldots, i_\ell \rangle \leftarrow \mathsf{Expand}_{r_1}(\mathsf{pw})$.

If $\sigma = 1$ then it computes a key by setting $\mathsf{key} \leftarrow \mathsf{Extract}_{r_2}(a_{i_1}, \ldots, a_{i_\ell}, \mathsf{pw})$, and othewise B chooses key uniformly at random.

Then B' activates the modified attacker $A'(\mathsf{key}, r_1, r_2, \vec{z}, \mathsf{pw})$, answering all of the queries using the "real solutions" that it knows. When A' halts and outputs a bit, B' chooses at random ℓ^* of the ℓ puzzles that are indexed by $\mathbf{i} = \mathsf{Expand}_{r_1}(\mathsf{pw})$. B' outputs the bit 1 if A' queried its oracle on any of these chosen puzzles and outputs the same bit as A' otherwise.

We denote $p_{\mathsf{real}}(B') \stackrel{\text{def}}{=} \Pr[B' \text{ outputs } 1|\sigma = 1]$ and $p_{\mathsf{rand}}(B') \stackrel{\text{def}}{=} \Pr[B' \text{ outputs } 1|\sigma = 0]$. It is easy to see that when $\sigma = 1$, then B outputs the bit one with exactly the same probability as B when it get "real solutions." Indeed, the only difference in this case is that B permutes its puzzles and B' does not, but since all the puzzles are drawn from the same distribution then the resulting distributions over puzzle-vectors are identical. Hence $p_{\mathsf{real}}(B') = p_{\mathsf{real}}(B)$.

It is also not hard to see that ehen $\sigma = 0$, the probability that B' outputs the bit one is at most δ away from the probability that B outputs the bit one when it gets "random solutions". The difference now (in addition to permuting the indexes) is that B computes the key as $\mathsf{key} \leftarrow \mathsf{Extract}_{r_2}(\cdots)$ while B' chooses it at random. But the ℓ^* random solutions have at least $\ell^*\mu = m'$ bits of min-entropy and $\mathsf{Extract}$ is a strong ranomdness extractor, so the statistical distance between these two distributions on the key is at most δ. Hence $p_{\mathsf{rand}}(B') \geq p_{\mathsf{rand}}(B) - \delta$.

We now consider the "bad event" in the run of B' in which A' had queried its H-oracle on any of the ℓ^* chosen puzzles, and moreover at the time of that query there were still un-chosen puzzles that are mapped to pw but were not yet queried. In other words, the complementing "good event" is that either the ℓ^* chosen puzzles were not queried at all, or at least they was queried after all the other puzzles that are indexed by $\mathbf{i} = \mathsf{Expand}_{r_1}(\mathsf{pw})$. Since we know that A' queries its H oracle on either ℓ or $\ell - \ell^*$ of the puzzles indexed by $\mathsf{Expand}_{r_1}(\mathsf{pw})$, then this "good event" can be described as B' choosing either the last ℓ^* H-queries among these puzzles (if A' queried them all) or the only ones that A' did not query (if A' left ℓ^* un-queried puzzles). Hence the probability of this "good event" is exactly $1/\binom{\ell}{\ell^*}$, regardless of the transcript of A'.

We complete the proof by observing that: (i) when the "bad event" happens then B outputs the bit 1 (since in this case A' queries on some of the chosen puzzles), (ii) the "bad event" is independent of the transcript of A' (since for any fixed transcript it always happens with probability exactly $1-(1/\binom{\ell}{\ell^*}))$, and (iii) the execution of B' conditioned on the "bad event" not happening is identical to the execution of A' (follows from the definitions of A', B': they both output 1 when all the puzzles indexed by $\mathsf{Expand}_{r_1}(\mathsf{pw})$ were queried and output whatever A outputs when only $\ell - \ell^*$ of them were queried and the "bad event" of B' does not happen). Thus we have

$$\begin{aligned} p_{\mathsf{real}}(B) - p_{\mathsf{rand}}(B) &\geq p_{\mathsf{real}}(B') - p_{\mathsf{rand}}(B') - \delta \\ &= \left(\left(1 - \frac{1}{\binom{\ell}{\ell^*}}\right) + \frac{p_{\mathsf{real}}(A')}{\binom{\ell}{\ell^*}}\right) - \left(\left(1 - \frac{1}{\binom{\ell}{\ell^*}}\right) + \frac{p_{\mathsf{rand}}(A')}{\binom{\ell}{\ell^*}}\right) - \delta \\ &= \frac{p_{\mathsf{real}}(A') - p_{\mathsf{rand}}(A')}{\binom{\ell}{\ell^*}} - \delta = \frac{\alpha(A) - \mathsf{acvr}(q, \mathsf{Expand}_{r_1}, D)}{\binom{\ell}{\ell^*}} - \delta \end{aligned}$$

By the hardness assumption on the puzzles, we know that the advantage of B is negligible in m, $p_{\mathsf{real}}(B) - p_{\mathsf{rand}}(B) = \mathsf{negl}(m)$, and it therefore follows that $\alpha(A) \leq \mathsf{acvr}(q, \mathsf{Expand}_{r_1}, D) + \delta\binom{\ell}{\ell^*} + \mathsf{negl}(m)$. □

Comment. We note that the solution-indistinguishability assumption and the use of strong extractor can be replaced with the more specific assumption that the output of $\mathsf{Extract}$ is pseudo-random. Namely, we can assume directly that given ℓ^* puzzles without their solution and $\ell - \ell^*$

puzzles with solution and given r_2 and key, an attacker cannot tell if the key was computed as $\mathsf{Extract}_{r_2}(a_1, a_2, \ldots)$ or was chosen at random. The proof under this (weaker) assumption is very similar to the proof of Lemma 6.

5 UC security analysis

In our second security analysis we use the UC security framework [Can01], and the presentation here assumes familiarity with this framework. We incorporate the human oracle H in the model by providing the adversary and the parties running the protocol with access to H. The environment is not given direct access to H; rather, it has access to H only via the adversary. This restriction represents the assumption that the puzzles used in an instance of the protocol are "local" to that instance, in the sense that they are generated within that instance, and furthermore the corresponding solutions are not affected by external events.

In addition, as in Section 4 we focus on conservative adversaries that ask H only on puzzles that were directly provided by the protocol. This technical restriction represents the "meta-assumption" that the puzzles are such that obtaining a solution for one puzzle does not help in solving a different puzzle. (Admittedly, this assumption may not always hold, and somewhat restricts the pool of potential implementations.)

The ideal password-based key generation and recovery functionality. We define the ideal functionality representing the security specification that we wish to obtain. The functionality, $\mathcal{F}_{\text{PKGR}}$, is presented in Figure 1. $\mathcal{F}_{\text{PKGR}}$ is parameterized by a dictionary D of possible passwords, the maximum number p of allowable password queries by the adversary, and the length m of the generated key. At the first activation $\mathcal{F}_{\text{PKGR}}$ expects the user to provide a password pw, along with a session identifier *sid*. (Formally, the *sid* may specify the identities of the user and the computer.) $\mathcal{F}_{\text{PKGR}}$ then generates a random m-bit key key and outputs it to the computer. Finally, it notifies the adversary that a key was generated. If pw is not in the dictionary then it also gives it to the adversary in full. (Formally this means that no security is guaranteed for passwords not in the dictionary. But note that our scheme does not depend on the actual dictionary, so we can always let the parameter D be the set of passwords that are "actually used by users".)

Next, whenever $\mathcal{F}_{\text{PKGR}}$ receives a password pw' together with a request to recover the key, it outputs the key to the computer only if pw' is the same as the stored password. This request may come from anyone, not only the legitimate user. Finally, $\mathcal{F}_{\text{PKGR}}$ answers up to p password guesses made by the adversary.

The main security guarantee of $\mathcal{F}_{\text{PKGR}}$ is that the adversary can make only p password guesses. If none of these guesses succeeds, then the key is indistinguishable from a truly random m-bit key. For reasonable values of m, this provides strong cryptographic security. Also, $\mathcal{F}_{\text{PKGR}}$ makes no mention of puzzles. Indeed, puzzles are treated as part of the implementation, geared toward limiting the number of password guesses.

Note that $\mathcal{F}_{\text{PKGR}}$ obtains the password directly from the environment. This formulation (which follows the formulation in [CHK$^+$05]) provides some strong guarantees. First, there is no a-priori assumption on the distribution from which the password is taken, as long as it is taken from D. In fact, this distribution may not even be efficiently generatable, since it may depend on the initial input of the environment. Second, we are guaranteed that in any scheme that realizes $\mathcal{F}_{\text{PKGR}}$ the local storage of the computer cannot be correlated in any way with the password *and the key.*

The fact that $\mathcal{F}_{\text{PKGR}}$ is parameterized by D, and furthermore provides no security for passwords not in D, is a limitation that comes from our simulation procedure. Ideally, we would like to

Functionality $\mathcal{F}_{\text{PKGR}}(D, p, m)$

1. At the first activation, receive from user U an input (`password`, sid, pw, aux), choose a random m-bit key key, output (sid, key) to U, notify the adversary that a key has been generated, and also send aux to the adversary. If $\mathsf{pw} \notin D$ then also send pw to the adversary.
2. When receiving input (`key-recovery`, sid, pw') from anyone, do: If $\mathsf{pw}' = \mathsf{pw}$ then output key to U. Else output an error message to U.
3. When receiving input (`password-query`, sid, pw') from the adversary, do: If the adversary already made p password queries then ignore. Else, if $\mathsf{pw}' = \mathsf{pw}$ then return key to the adversary. Else return `Wrong-Guess` to the adversary.

Figure 1: The ideal functionality $\mathcal{F}_{\text{PKGR}}$, parameterized by a dictionary D, the maximum number p of password queries, and the length m of the generated key.

guarantee that $\mathcal{F}_{\text{PKGR}}$ does not depend on D at all and provides security for any password string. Realizing such a functionality is an interesting future challenge.

Invertibility of Extract. In order to show that the scheme realizes $\mathcal{F}_{\text{PKGR}}$ we need to make an additional "invertibility" assumption on function Extract with respect to solutions of puzzles: What we need is that given an m-bit key key and given $\ell - \ell^*$ solution to ℓ puzzles, one can efficiently compute ℓ^* plausible solutions to the remaining puzzles that would map the entire solution vector to the given key, $\mathsf{Extract}(\ldots) = \mathsf{key}$.

Below let (G, H) be a puzzle system with μ bits of pseudo-entropy, where the right solution for each puzzle z is indistinguishable from a solution drawn from $R(z)$, let f_r be a randomized function, and let $\ell \geq \ell^*$ be integers.

Definition 6 *We say that f is* strongly invertible w.r.t. the puzzle system (G, H) *and the parameters ℓ, ℓ^* if there exists an efficient inversion algorithm I that given randomness r, key* key*, any string* pw*, ℓ puzzles $z_1, ..., z_\ell$ generated via G, and a set of $\ell - \ell^*$ solutions $a_{\ell^*+1}, ..., a_\ell$ such that $a_i \in R(z_i)$, outputs values $a_1, ..., a_{\ell^*}$ such that $a_i \in R(z_i)$ and $f_r(a_1, ..., a_\ell, \mathsf{pw},) = \mathsf{key}$.*

Furthermore, for random $r, \mathsf{key}, z_1, \ldots, z_\ell$ and any pw *and $a_{\ell^*+1}, ..., a_\ell$, the output of I is indistinguishable from sampling at random $a_i \in R(z_i)$ for $i = 1, \ldots, \ell^*$ subject to the constraint $f_r(a_1, ..., a_\ell, \mathsf{pw},) = \mathsf{key}$.*

If the distributions $R(z)$ are efficiently sampleable given z and if m is small enough (so that 2^m is polynomial in the relevant efficiency parameters) then I can simply sample the solutions $a_1, ..., a_{\ell^*}$ from the appropriate distributions until it find a solution vector that match the given key. Alternatively, if we have a puzzle-system for which $R(z) = \{0,1\}^k$ for some k and Extract is a linear function (e.g., a linear universal hash function) then inversion is possible even for large m via linearity.

UC-security of our scheme. Consider the scheme from Section 3 with parameters n, ℓ, and m, and let D be the dictionary used by the user. Assume that (G, H) is a puzzle system with μ bits of pseudo-entropy, assume that Extract extracts m bits that are negligibly close to uniform from any distribution with $m' > m$ bits of min-entropy, and let $\ell^* = \lceil m'/\mu \rceil$. Finally assume that Extract is strongly invertible w.r.t. (G, H) with parameters ℓ, ℓ^*, and let $p = p(q)$ be a large enough bound such that $\Pr_{r1}[\mathsf{acvrSet}(q, \mathsf{Expand}_{r1}, D) > p/|D|] = \mathsf{negl}(m)$. Then we have

Lemma 7 *Under the conditions above, the scheme from Section 3 UC-realizes $\mathcal{F}_{\text{PKGR}}(D, p, m)$ relative to conservative adversaries that makes at most q queries to H.*

Proof: Let $\mathcal{A}$ be an adversary that interacts with the scheme. Recall that $\mathcal{A}$ queries H at most q times, and that all the queries are puzzles that appear explicitly in the information obtained from the scheme itself.

We construct an adversary (simulator) $\mathcal{S}$ that interacts with $\mathcal{F}_{\text{PKGR}}$ such that any environment can tell only with negligible probability whether it is interacting with the scheme and $\mathcal{A}$ or with $\mathcal{F}_{\text{PKGR}}$ and $\mathcal{S}$. $\mathcal{S}$ invokes a simulated copy of $\mathcal{A}$, and proceeds as follows:

1. Throughout, all inputs from the environment are forwarded to $\mathcal{A}$, and all outputs generated by $\mathcal{A}$ for its environment are forwarded to $\mathcal{S}$'s environment.

2. When notified by $\mathcal{F}_{\text{PKGR}}$ that a key has been generated (and given the auxiliary information aux), $\mathcal{S}$ hands a simulated contents of the memory to $\mathcal{A}$. That is, $\mathcal{S}$ runs the puzzle generator $G(1^m, \mathsf{aux})$ n times on independent random inputs and hands the generated puzzles $z_1, ..., z_n$ to $\mathcal{A}$. (Recall that at this point, the environment obtains a random generated key key from $\mathcal{F}_{\text{PKGR}}$, via the user U.) In addition $\mathcal{S}$ gives $\mathcal{A}$ the random seeds r_1, r_2 for $\mathsf{Expand}, \mathsf{Extract}$.

3. When $\mathcal{A}$ attempts to query H on a puzzle z_i, $\mathcal{S}$ does:
   (a) $\mathcal{S}$ queries H on z_i and obtains the answer $a_i = H(z_i)$.
   (b) In addition, $\mathcal{S}$ keeps a record of all the puzzles $z_{i_1}, \ldots, z_{i_\omega}$ that were queried up to this point. It then finds the set C of passwords in D that are almost covered by $i_1, ..., i_\omega$ (namely, the passwords s such that at most ℓ^* puzzles in $\mathsf{Expand}(s)$ were not queried) and queries $\mathcal{F}_{\text{PKGR}}$ on all the passwords in C that were not yet queried. Note that since $\omega \leq q$, we have that the overall number of passwords queried by $\mathcal{S}$ is at most $\mathsf{acvrSet}(q, \mathsf{Expand}_{r1}, D) \cdot |D|$.
   (c) If none of the password queries succeed, then $\mathcal{S}$ returns a_i to $\mathcal{A}$.
   (d) If one of the queries succeeds and $\mathcal{S}$ obtains the key key, then $\mathcal{S}$ recalls the previous $\ell - \ell^*$ answers $a_{i_1}, \ldots, a_{i_{\ell-\ell^*}}$ that correspond to the correct password pw, and runs the inversion algorithm I from Definition 6 to choose values $a_{\ell-\ell^*+1}, ..., a_\ell$ such that $\mathsf{Extract}_r(a_{i_1}, \ldots, a_{i_\ell}, \mathsf{pw}) = \mathsf{key}$. From now on, each $\mathcal{S}$ answers each query for a puzzle z_i where $i \in \mathsf{Expand}(\mathsf{pw})$ with the solution a_i.

Before asserting the validity of $\mathcal{S}$, we analyze its complexity. The only potentially hard part is in determining the set of passwords covered by a given index set $i_1, ..., i_\omega$. If the size of the dictionary D is sufficiently small (formally, polynomial in the security parameter), then $\mathcal{S}$ can simply try all passwords in D. In case that D is larger (say, if the user actually uses a cryptographically strong key as a password), we assume that D and Expand have enough structure that allows determining the desired set in an efficient way.

Validity of $\mathcal{S}$. Validity of $\mathcal{S}$ is asserted via a reduction to the solution indistinguishability of the puzzle system (G, H). The reduction is very similar to the reduction in the proof of Lemma 6.

Let E be an environment that has advantage α in distinguishing the protocol from above (running against the dummy adversary) from the simulator with $\mathcal{F}_{\text{PKGR}}$. We use E to distinguish between real and random in (G^{ℓ^*}, H^{ℓ^*}). The distinguisher D gets $\langle(z_1, a_1), \ldots, (z_{\ell^*}, a_{\ell^*}), (z_{\ell^*+1}, a_{\ell^*+1}, a'_{\ell^*+1}), \ldots, (z_n, a_n, a'_n)\rangle$ (and auxilary input aux), and it activates $E(\mathsf{aux})$, playing for it the entire execusion as follows: when E activates the user U with password pw, D chooses r_1, r_2, computes $\mathbf{i} = \langle i_1, \ldots, i_\ell\rangle \leftarrow \mathsf{Expand}_{r_1}(\mathsf{pw})$ and permutaed the puzzles randomly subject to the contsraint that the first ℓ^* puzzles are mapped to indexes in $\mathbf{i}$. It then computes $\mathsf{key} \leftarrow \mathsf{Expand}_{r_1}(a_{i_1}, \ldots, a_{i_\ell})$ (with indexing relative to the permuted puzzles) and return key to E together with the permuted vector of puzzles $\vec{z}$.

When E asks (via the dummy adversary) about the solution to some puzzle z then D replies as follows: If it has two solutions a, a' for z then it replies with the real solution a. If E asks about any of the puzzles for which D knows only one solution *and if at that time there are other puzzles that are mapped to* pw *and for which D knows both solutions bit they were still not queried* then D aborts, outputting the bit one. If E asks about the puzzles for which D knows just one solution only after it had asked about all the other puzzles that are mapped to pw then D replies with the solution that it knows. If D does not abort and E halts with some output, then D outputs the same.

The analysis of the advantage of D is nearly identical to the analysis of the reduction B from Lemma 6 and is omitted. $\square$

6 A Concrete Examples

We describe a concrete instantiation of our scheme for a "medium security" application. The example builds on the numeric example in Section 3.2. Consider trying to get a 64-bit key using our scheme, while relying on a CAPTCHA system whose assumed hardness is (say) 16 bits of pseudo-entropy per instance. In terms of our analysis, we therefore have the parameters $m = 64$ and $\mu = 16$ (so $\ell^* = 64/16 = 4$). Assuming that each puzzle takes about 4KB to encode (which is the case for common CAPTCHAs) and restricting ourselves to number of puzzles that fit on a single 4.7GB DVD-R, we would like to store about $n = 2^{20}$ puzzles.

We assume that the odds of the attacker guessing a weak password are one in a million (per guess) so we assume the dictionary size is $|D| = 2^{20}$. Also, we would like the legitimate user to solve no more than $\ell = 8$ puzzles to access the encrypted data, and we assume that the attacker cannot get a human to solve for it more than $q = 10^4$ puzzles. (This seems like a safe assumption in most settings, but maybe not all of them.) In Section 3.2 it is shown that with this setting of the parameters, and when modeling Expand as a random function, we should expect the cover-number to be no more than 0.7%. Namely, an attacker that makes up to 10,000 puzzle-solving queries has at most 0.7% chance of "almost covering" the actual password. (Note that the obvious way of "almost covering" passwords yields success probability of 0.25%.)

A simple heuristic instantiation of the scheme for this setting of the parameters would be to use a cryptographic hash function to implement both the Expand and $\mathsf{Extract}$ functions. More specifically, to generate the key, choose (say) two 128-bit salt values r_1 and r_2, get the user's password pw and compute $h \leftarrow$HMAC-SHA1$_{r_1}(\mathsf{pw})$. Parse the 160 bits of h as eight 20-bit indexes $i_1, \ldots, i_8 \in [2^{20}]$. Generate 2^{20} CAPTCHAs and store then together with r_1, r_2 on the disk (or on a DVD-R). Then obtain the solutions to the eight relevant puzzles $a_{i_j} = H(z_{i_j}),\ j = 1 \ldots 8$, compute $h' \leftarrow$HMAC-SHA1$_{r_2}(a_{i_1}|\ldots|a_{i_8}|\mathsf{pw})$, and take as many bits of h' as needed for the key. To make use of the common practice for slowing down brute-force attacks, replace SHA1 with the iterated function SHA1^k for some reasonable k, say $k = 65536$.

Although this scheme allows the key to be longer than 64 bits, the above analysis only shows that with probability more than 99.3% we get "64-bit strength". (One can similarly calculate the probability of getting "80-bit strength" which happens when the attacker misses five of the eight puzzles that are mapped to the user's password, or the probability of getting "128-bit strength" when the attacker misses all the puzzles, etc.)

7 Future Work

This work investigates a new approach for deriving cryptographic keys from human passwords, improving the resistance against off-line dictionary attacks by having the user solve some puzzles and using the solutions in the key-generation process. Still, the analysis is quite preliminary and many questions remain unanswered, regarding constructions, modeling, and analysis. Below we list a few of these questions.

Building Puzzles. Perhaps a first challenge is to actually construct puzzle systems that are useful for schemes such as the one described in this paper. Of particular interest for our scheme would be puzzles that remain hard even when the attacker knows the randomness that was used to generate them. (As mentioned above, using such puzzle-systems we can forgo storing any puzzles on the disk, instead using the value $\mathsf{Expand}(\mathsf{pw})$ as randomness for the puzzle generation system.)

As discussed in the introduction, another important property of puzzles is "non-malleability", namely preventing the adversary from using the solution to one puzzle to solve other puzzles. This question is interesting both on the level of designing puzzles that will make it hard for the adversary to "maul" them while maintaining solvability, and on the level of mathematical formalization. On the design level, a potentially interesting approach might be to use watermarking techniques to embed in the puzzle a message to the human reader that urges to not solve the puzzle outside a certain context.

Improving the scheme. One obvious challenge is to find different constructions that will improve various characteristics of the current scheme (e.g., use less storage, improve security, etc.) An immediate question is to have an efficient explicit construction for Expand that provably has a small expected cover number (maybe by extending the work of Alon et al. [ADM$^+$99]). Another direction is to split the key-generation process to two parts, where the first part generates a large amount of storage which is the same for all users and the second adds a small amount of user-specific storage that may depend on the password or on other information provided by the user. Another direction is to have a more interactive key recovery process, where the user's answer to the current question effects the questions that it is asked next.

Improving the analysis. The analysis can be improved in a number of ways. First, the hardness assumption on puzzles may be reduced to allow for non-negligible distinguishing probability between the real solution and a random one. Furthermore, one might want to assume only that *computing* the correct answer is hard. Another immediate set of goals is to improve the current analysis (especially the one in the UC framework), and to find ways to relate the game-based security and the UC security notions.

Also, it would be nice to be able to formally model and argue about non-conservative adversaries, namely adversaries that query the human oracle on puzzles different than the ones used in the scheme. What security properties can be guaranteed against such adversaries? Another plausible extension of our security model is to consider an attacker that can modify the storage on the disk and then watch the output of the key-recovery procedure with this modified storage. (This seems related to the issue of puzzle malleability that we mentioned above.)

Also, it may be useful to have an algorithm to compute (or bound) the cover number of a given Expand function with respect to a given dictionary. This seems harder than computing/bounding the expansion of a graph, since here we need to compute/bound the expansion of any subgraph of a given degree.

A more speculative research direction is to try and obtain reasonable general models for human users (or even human attackers) in systems such as the ones described in this work.

References

[AC88] Noga Alon and Fan R.K. Chung, Explicit constructions of linear sized tolerant networks. *Discrete Mathematics*, 72:15-19, 1988.

[ADM+99] Noga Alon, Martin Dietzfelbinger, Peter Bro Miltersen, Erez Petrank, Gábor Tardos. Linear Hash Functions. J. ACM 46(5): 667–683. 1999.

[BPR00] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. *Advances in Cryptology – Eurocrypt 2000*, LNCS vol. 1807, Springer-Verlag, pp. 139–155, 2000.

[BR93] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. *Advances in Cryptology – Crypto 1993*, LNCS vol. 773, Springer-Verlag, pp. 232–249, 1993.

[Can01] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. *42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, IEEE, pp. 136–145, 2001.

[CHK+05] R. Canetti, S. Halevi, J. Katz, Y. Lindell and P. MacKenzie. Universally composable password-based key exchange. *Advances in Cryptology – Eurocrypt 2005*, LNCS vol. 3494, Springer-Verlag, pp. 404–421, 2005.

[DDN00] Danny Dolev, Cynthia Dwork, Moni Naor. Non-malleable Cryptography. *SIAM J. Comput.* 30(2): 391-437 (2000)

[DPPU88] Cynthia Dwork, David Peleg, Nicholas Pippenger, Eli Upfal: Fault Tolerance in Networks of Bounded Degree. *SIAM J. Comput.* 17(5): 975-988 (1988)

[DN92] Cynthia Dwork, Moni Naor. Pricing via Processing or Combating Junk Mail. *Advances in Cryptology – Crypto 1992*, LNCS vol. 740, Springer-Verlag, pp. 139–147, 1992.

[Kal00] Burt Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898 `http://www.ietf.org/rfc/rfc2898.txt`. September 2000

[Naor96] Moni Naor. Verification of a human in the loop or identification via the Turing test. Manuscript, available on-line from `http://www.wisdom.weizmann.ac.il/~naor/PAPERS/human_abs.html`, September 1996.

[NP97] Moni Naor and Benny Pinkas. Visual Authentication and Identification *Advances in Cryptology – Crypto 1997*, LNCS vol. 1294, Springer-Verlag, pp. 322–336, 1997

[NZ96] Noam Nisan, David Zuckerman. Randomness is Linear in Space. *J. Comput. Syst. Sci.* 52(1): 43–52. 1996.

[PS02] Benny Pinkas and Tomas Sander. Securing passwords against dictionary attacks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 161–170, Washington, DC, USA, November 2002. ACM Press.

[SS04] Adam Stubblefield and Dan Simon. Inkblot Authentication. Microsoft Research Technical report MSR-TR-2004-85.

[U94] Eli Upfal Tolerating a Linear Number of Faults in Networks of Bounded Degree *Inf. Comput.* 115(2): 312-320 (1994). (Earlier version in PODC'92.)

[vAB+03] Luis von Ahn, Manuel Blum, Nicholas Hopper, and John Langford. CAPTCHA: Using hard AI problems for security. In *Advances in Cryptology – EUROCRYPT '2003*, volume 2656 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 2003.

A Possible implementations of puzzles

The notion of puzzles used in our scheme is rather relaxed, and consequently has a large number of potential implementations. One set of implementations are the existing and potential implementations of CAPTCHAs, e.g. the ones described in [Naor96, vAB+03]. These implementations, however, require that the solutions to puzzles are uniquely defined and generatable together with the puzzles. This seems like a strong limitation on implementations. Another set of implementations are the potential implementations of Inkblots, described in [SS04]. These implementations, however, are required to be "private", in the sense that the solution given by one human is unpredictable (and, in fact, pseudorandom) even to other humans. Again, this seems to be a strong limitation on implementations.

Below we list some potential approaches for implementation that are not included in any of these classes. That is, the answers to these puzzles are somewhat subjective and not necessarily unique. Still, answers of different individuals may be similar. As pointed out in [NP97] in a different context, validity of these proposals should be evaluated by testing on human individuals.

Personal ranking: Puzzles may include pictures of different persons, to be ranked by coolness, or age, or taste in clothing. Alternatively, puzzles may include different pictures or descriptions of food, to be ranked by tastiness, spiciness, etc.. Alternatively, a puzzle may depict several randomly generated drawings to be ranked by personal liking, an audio puzzle may sound several short melodies to be ranked by liking, etc.

Face recognition: Puzzles may include pictures of several faces, and the question is which of these faces most resemble people known to the user (e.g., parents or other family members). Furthermore, puzzles can be "personalized" by asking the user to provide, at system set-up, pictures of close family members. These pictures might then be mixed with random pictures to generate puzzles. (This proposal takes advantage of the fact that recognizing faces is one of the most highly specialized visual capabilities of humans.)

Personal clustering: Puzzles may include a bunch of various unrelated objects, and the question is which three objects "go together" the best, or are the most "closely related" or "look alike". The objects can be people, household items, cartoons, or a mix of all categories. (This implementation approach was proposed by Ronitt Rubinfeld.)

Imaginative inferring: Puzzles may portray a scene and ask questions about what happened a minute ago, or what will happen in a minute. Alternatively, questions can be asked regarding what happens outside the borders of the picture.

Personal association: Puzzles may depict an object (e.g., a person) and ask which familiar objects (persons) does the object in the picture remind.

Finally, we note that to prevent "mauling attacks," the puzzle generator must make sure that randomly generated puzzles are "different enough" from each other so that a solution to one randomly generated puzzle will not help in solving another randomly generated puzzle.