A practical, perfectly secure password scheme in the bounded retrieval model

Moses Liskov*

August 30, 2017

Abstract

In this paper, we present a practical password scheme due to Spilman, which is perfectly secure in the bounded retrieval model, assuming ideal hash functions. The construction is based on a hash-like function computed by a third party "facilitator". The facilitator is trusted, and security derives from the facilitator's long random secret, although the adversary is assumed to be able to retrieve a large fraction of that secret.

Unlike the traditional "salted and hashed password" approach, this scheme is secure against an adversary capable of performing brute force dictionary attacks offline. The key security property for the facilitator function is a form of uncloneability, that prevents the adversary from calculating function values offline.

1 Introduction

Advances in computer technology over the past two decades have greatly increased the capacity for storage of large amounts of data. These advances have the potential to improve cryptographic techniques by leveraging the distinction between the storage of large amounts of data and the *communication* of large amounts of data.

The "bounded retrieval model" (BRM) of DiCrescenzo, Lipton, and Walfish [2] limits the adversary to exposing a limited amount of private data, plus an unlimited amount of computation. The BRM was influential towards the notion of leakage-resilient cryptography, which led to a proliferation of interesting results.

Our contributions. In this paper, we present a practical password scheme due to Spilman [4], which is perfectly secure in the bounded retrieval model, assuming ideal hash functions. The construction is based on a hash-like function

*`mliskov@mitre.org`. Copyright (c) 2017 The MITRE Corporation. ALL RIGHTS RESERVED. Approved for Public Release; Distribution Unlimited. Case Number 17-2748.

computed by a third party "facilitator". The facilitator is trusted, and security derives from the facilitator's long random secret, although the adversary is assumed to be able to retrieve a large fraction of that secret.

Password security. Servers typically do not store the passwords of users in the clear, because this might compromise the security of the accounts. Instead, servers frequently store passwords hashed and salted. That is, for each user account, a pair (s_i, y_i) is stored, where s_i is an arbitrary random value, and y_i is the result of hashing the user's password x_i with s_i. Passwords can be checked by recomputing the hash. An attacker that obtains (s_i, y_i) without knowing the password can recover the password through a brute force dictionary attack. Although this may require a large amount of computation, low-entropy passwords can be recovered after some effort if the attack can obtain the (S, C) pair and recover the password without attempting interactive logins, a so-called off-line attack.

The central idea of Spilman's construction [4] is to provide a service that calculates a function whose behavior cannot be predicted even if an adversary exfiltrates a large portion of a very large secret key. This function, rather than an offline-computable function such as a hash function, is used to calculate the check value.

Uncloneable functions A *physically uncloneable function* is a hardware device based on some physical manifestation of randomness that can calculate a function with some degree of unpredictability in its output. For instance, the device might include a block of resin with impurities mixed in, and shine a laser through the resin in various trajectories and measure the absorption. Measuring the absorption along the same trajectories will give more or less consistent results, but without exactly copying the block of resin, no attacker would be able to give more outputs of the function than they had queried themselves.

The connection between uncloneable functions and password security is simple: instead of using a cryptographic hash function to calculate the check value, use the uncloneable function. An attacker that has no further access to the uncloneable function will learn nothing from seeing the check value, unless the attacker has by chance already learned an input that produces that check value.

This idea could be used with actual PUFs, but the absolute uncloneability of the PUFs is problematic, because it means that password-checking cannot be replicated. Any replica must have access to the PUF, and there is only one. This results in a scaling problem, and also, the PUF is a single point of failure. If it were to be maliciously or accidentally destroyed, the entire password-checking system would become indefinitely unavailable.

In contrast, we consider the notion of a *random uncloneable function*, a construction that makes use of an ideal hash function and a large random secret (the "corpus") to implement a function that is indifferentiable from a random function in the bounded retrieval model. This kind of uncloneable function *can* be replicated by honest parties, by paying the cost of fully replicating the corpus.

1.1 Prior work.

Although this paper relies heavily on the algorithmic constructions of Spilman [4] (although the ones here are simplified in certain ways from Spilman's version), its contribution is in the theoretical treatment of the ideas. Spilman describes a construction but does not provide clear definitions of success or proofs of correctness. We define the notion of a random uncloneable function, prove a construction satisfies it, and also prove that a random uncloneable function provides offline password security.

The notion of password security in the bounded retrieval model was the focus of the seminal paper of DiCrescenzo, Lipton and Walfish [2]. However, our results are substantially different, in that the password scheme here is practical and achieves security for arbitrarily large absolute retrieval bounds.

2 Definitions

Oracle Turing machines. The definitions in this paper largely involve Turing machines with one or more oracles. The number of queries an oracle Turing machine makes to each of its oracles is the way in which the power of the machine is understood, since we make no assumptions about time or space complexity. It will be convenient to refer to oracle Turing machines that are implicitly regulated in their number of queries. A $(q_1, \ldots, q_m)$-bounded oracle Turing machine (or BOTM for short) is an oracle Turing machine with m distinct oracles, the ith of which may be queried at most q_i times.

Random uncloneable functions. Our notion of a random uncloneable function transforms a hash-like random oracle into a random uncloneable function in the BRM.

This is a reduction of random oracles, following the ideas of Maurer et al. [3]. The same notion was used by Coron et al. [1] to construct an ideal hash function from an ideal compression function. We do something similar. Rather than extend the domain of a random function (from $\{0,1\}^{2n}$ to $\{0,1\}^*$, as Coron et al. do), we achieve uncloneability.

We give the formal definition of indifferentiability [3]:

Definition 2.1. *A Turing Machine C with oracle access to an oracle $\mathcal{H}$ is said to be (q_1, q_2, ϵ)-indifferentiable from $\mathcal{F}$ if there exists a PPT simulator S such that for all (q_1, q_2)-BOTMs D,*

$$|\mathsf{Pr}[D^{C,\mathcal{H}} = 1] - \mathsf{Pr}[D^{\mathcal{F},S^{\mathcal{F}}} = 1]| < \epsilon$$

Note that we could regard the above definition as applying to families of constructions C, parameterized by some security parameter. This statement tends to be of interest for families only if ϵ is a negligible function in k.

Here, $S^{\mathcal{F}}$ means that S is a Turing Machine with oracle access to $\mathcal{F}$.

If we take oracles to be stateful, this definition could be used for a construction C that has some sort of private value, generated on its first invocation, and maintained in C's memory thereafter. However, our aim is to define indifferentiability when the adversary (distinguisher) has access to some portion of C's private data. This motivates the following definition:

Definition 2.2. *Let C be a Oracle Turing Machine which takes input $\vec{c} = c_1, \ldots, c_s$ where each $c_i \in \{0,1\}^k$, and which maintains no long-term state.*

We say $C^{\mathcal{H}}$ is $(q_1, q_2, q_3, \epsilon)$-indifferentiable from $\mathcal{F}$ in the adaptive bounded retrieval model (ABRM) if there exists a simulator S such that for all (q_1, q_2, q_3)-BOTMs D,

$$|\mathsf{Pr}[\vec{c} \leftarrow \{0,1\}^{ks}; D^{C(\vec{c}),\mathcal{H},R(\vec{c})} = 1] - \mathsf{Pr}[D^{\mathcal{F},S^{\mathcal{F}}(0,\cdot),S^{\mathcal{F}}(1,\cdot)} = 1]| < \epsilon$$

Here, $S(0,x)$ attempts to emulate $\mathcal{H}(x)$ and $S(1,j)$ attempts to emulate $R(j)$, and we assume that all invocations of S maintain state.

We can also define the above notion in a non-adaptive model, where the adversary receives q_3 locations of $\vec{c}$ that must be chosen all in advance. However, the construction in this paper satisfies the stronger definition.

2.1 The construction C

The construction we present is based on the Blind Hash algorithm of Spilman [4].

The construction is parametrized by a security parameter k and by a corpus size s. We assume that the corpus consists of s random strings of length k, referred to as $c_1, \ldots, c_s$. We assume also that $\mathcal{H}$ is a random oracle mapping $\{0,1\}^* \to \{0,1\}^k$.

Finally, we assume that Loc is a function that for a uniform random k-bit input produces a uniform random output in the range $[1, s]$. For instance, if $s = 2^l$, Loc could simply truncate its input.

When j is a number, we write $\mathbf{j}$ to indicate a $\lfloor \log_2(k+1) \rfloor$-bit binary string that represents j.

Let x be the input to C. We calculate k locations $l_1, \ldots, l_k$, where $l_j = \mathsf{Loc}(\mathcal{H}(\mathbf{j}||x))$. We then concatenate the corresponding corpus contents and hash this to produce the final result:

$$C(x) = \mathcal{H}(\mathbf{0}||c_{l_1}|| \ldots ||c_{l_k}).$$

Theorem 2.3. *$C^{\mathcal{H}}$ is $(q_1, q_2, q_3, \epsilon)$-indifferentiable from random, for*

$$\epsilon = \frac{s(s+1) + 2q_2^2/(k - k\log_2 s + 2)^2}{2^{k+1}}.$$

Proof. To prove this, we must describe the simulator S that emulates random $\mathcal{H}$ outputs with only oracle access to the $\mathcal{F}$ it must match.

The general strategy for S is to randomly select $\mathcal{H}$ outputs, except when those outputs are known or likely to be used as $\mathcal{F}$ outputs. Specifically, let $\alpha = \lceil k/\log_2 s \rceil$, and answer queries as follows:

- On input $(0, x)$ where $S(x)$ is defined, return $S(x)$. On input $(1, i)$ where c_i is defined, return c_i.
- On input $(1, i)$ where c_i is not defined, pick a random block distinct from all previously defined corpus blocks, and define c_i as that block and retutn c_i.
- On input $(0, x)$ where $S(x)$ is not defined:
  1. On input $(0, \mathbf{j}||x)$ with $1 \leq j \leq k$, where fewer than $k - \alpha$ of $S(\mathbf{1}||x), \ldots, S(\mathbf{k}||x)$ are defined, define $S(\mathbf{j}||x)$ at random and return it.
  2. On input $(0, \mathbf{j}||x)$ where at least $k - \alpha$ of $S(\mathbf{1}||x), \ldots, S(\mathbf{k}||x)$ are defined, define all the undefined ones at random and return $S(\mathbf{j}||x)$.
  3. On input $(0, \mathbf{0}||v_1|| \ldots ||v_k)$ where each v_i is of length k, such that for some x and all $1 \leq j \leq k$: (1) $r_j = S(\mathbf{j}||x)$ is defined, and (2) $c_{\mathsf{Loc}(r_j)}$ is defined and is equal to v_j, define $S(\mathbf{0}||v_1|| \ldots ||v_k) = \mathcal{F}(x)$ and return $\mathcal{F}(x)$.
  4. Otherwise, on input $(0, x)$, define $S(x)$ to be a random value and return $S(x)$.

Note that in the "real" experiment, where all corpus blocks are chosen at random in advance, the probability that any pair of corpus blocks are identical is bounded by $s(s+1)/2^{k+1}$. Therefore, if the adversary has avantage ϵ, the adversary must have advantage at least $\epsilon - s(s+1)/2^{k+1}$ conditioned on the fact that the corpus consists only of unique blocks.

When we define $S(\mathbf{0}||v_1|| \ldots ||v_k)$ at random, we are taking a risk. If the distinguisher manages to find some x for which $c_{\mathsf{Loc}(S(\mathbf{j}||x))} = v_j$ for all j, the probability that $F(x)$ is the random output we chose is very small. Thus, we must argue that the probability that this event occurs is small.

The bad event occurs when $S(\mathbf{0}||v_1|| \ldots ||v_k)$ has been defined at random, and at some later point, there is an x such that for each $1 \leq j \leq k$, (1) $S(\mathbf{j}||x)$ has been defined, (2) $c_{\mathsf{Loc}(S(\mathbf{j}||x))}$ has been defined, and (3) $v_j = c_{\mathsf{Loc}(S(\mathbf{j}))}$.

A one-to-one correspondance can be drawn between all runs of the adversary against the $C^{\mathcal{H}}, \mathcal{H}, R$ oracles and runs of the adversary against the simulator in which bad does not occur. Therefore, the adversary's advantage in distinguishing can be limited to the probability that bad occurs.

If (1) happens at a later point for some specific x, then the last α of the $S(\mathbf{j}||x)$ are defined at random simultaneously. For any given j, the probability that conditions (2) and (3) are met is either $1/s$ if v_j is equal to some defined corpus block, or $1/2^k$ otherwise. Thus, the overall probability can be bounded by $s^{-\alpha}$. Note that since $\alpha > k/\log_2 s$, $\alpha \log_2 s > k$, so $s^\alpha = 2^{\alpha \log_2 s} > 2^k$. Therefore, we can bound the probability in this case by 2^{-k}.

If (1) is already true of every j when the query is made for some particular x, then (2) and (3) can only become true at a later point if at least once, an undefined corpus block is randomly chosen to be equal to the specific v_j value present in the query. This probability can be bounded by 2^{-k}.

To obtain the overall probability that bad occurs, we need a bound on the number of queries that assign $S(\mathbf{0}||v_1||\dots||v_k)$ at random, and on the number of x that could meet the three conditions for that query. For each pair, a query is possible with probability at most 2^{-k}. The number of pairs is maximized when the number of the former type of query is equal to the number of such x. One of each can be created for every $k - \alpha + 2$ queries, so the total probability of bad occurring may be bounded by $(q_2/(k-\alpha+2))^2$.

Therefore, the probability that the adversary distinguishes the real experiment from the simulator is at most $(q_2/(k-\alpha+2))^2/2^k + s(s+1)/2^{k+1} = \frac{s(s+1)+2q_2^2/(k-\alpha+2)^2}{2^{k+1}}$. □

Relating uncloneability and indifferentiability. Suppose a function F is drawn from some family $\mathcal{F}$ of functions, and an adversary is given oracle access to F. If this adversary is able to "learn" the function F, so that inputs and outputs of F may be determined without further oracle access to F, the adversary has succeeded in cloning the function F.

The classical notion of uncloneability is tested by challenging the attacker to compute $F(x)$ *with high probability* where x is chosen randomly from some set. Equivlently, the attacker could be asked to compute $F(x)$ for every x in some set of random challenges, with non-negligible probability.

Indifferentiability of F from a random function is a stronger property than this classical notion of uncloneability. Obviously, if a function can be cloned, the ability of an attacker to predict its outputs is a test that can distinguish between the true function and a random one.

In this sense, indifferentiability from random is a strong form of uncloneability, implying the classical notion of uncloneability. The property our password scheme relies on relates to the one-wayness of the function F, which is also implied by its indifferentiability from a random function.

3 Our password scheme

We imagine a scenario in which k honest users create random passwords $(x_1, \dots, x_k)$ with a server, with the help of a third party we call the facilitator, which implements the uncloneable random function. We model the passwords as random strings of length e, to emulate the notion that passwords are of low or medium entropy.

The facilitator has a large private key called the *corpus*, whose size is s blocks of k bits each. The facilitator operates a public service where any interested party may request the output of the uncloneable function on an input of their choice.

$\mathsf{Adv}_{\mathsf{pwd}}(A_1, A_2, C, \mathcal{H}) =$
$\mathsf{Pr}[\vec{c} \leftarrow \{0,1\}^{ks}; \alpha \leftarrow A_1^{C^{\mathcal{H}}(\vec{c}), \mathcal{H}, R(\vec{c})}(k);$
$\quad x_1 \leftarrow \{0,1\}^e; s_1 \leftarrow \{0,1\}^k; y_1 \leftarrow C^{\mathcal{H}}(\vec{c}, x_1 || s_1);$
$\quad \ldots$
$\quad x_k \leftarrow \{0,1\}^e; s_k \leftarrow \{0,1\}^k; y_k \leftarrow C^{\mathcal{H}}(\vec{c}, x_k || s_k);$
$\quad (x', j) \leftarrow A_2^{C^{\mathcal{H}}(\vec{c}), \mathcal{H}, R(x_1, \ldots, x_k)}(\alpha, y_1, \ldots, y, s_1, \ldots, s_k):$
$\quad C^{\mathcal{H}}(x' || s_j) = y_j] - \frac{q_1+1}{2^e}.$

The attack model we imagine is partially adaptive and partially non-adaptive. Specifically, security is described based on a password-guessing game that runs in two phases. In the first phase, the adversary is allowed to compromise up to q_3 blocks of the corpus, and make queries to the construction C as well as to the hash function $\mathcal{H}$. At the end of the first phase, the passwords $x_1, \ldots, x_k$ and salts $s_1, \ldots, s_k$ are chosen at random, and the check values $y_1, \ldots, y_k$ are calculated. In the second phase, the adversary may make additional queries to C and to $\mathcal{H}$, and may also request that up to $k-1$ of the passwords be revealed. However, the adversary may retrieve no additional blocks of the corpus in the second phase. At the conclusion of the second phase, the adversary guesses a specific non-revealed password.

It is possible that a stronger, fully adaptive notion of security can be achieved, but such a notion is a bit absurd. In order to continue retrieving blocks of the corpus, the adversary would have to maintain an infiltration into the facilitator's system. However, a real-world adversary able to do this could run a password search inside the facilitator's system, accessing but not exfiltrating any blocks of the corpus, and simply exfiltrating the final password.

Therefore, we imagine that no active presence inside the facilitator exists, but the adversary may have already exfiltrated a substantial amount of the corpus.

3.1 Definition

In our definition, we model the adversary as a BOTM with two explicit parts: A_1, which may perform corpus block retrieval, and A_2, which may not. A_1 takes as input only the security parameter k, and outputs only a state α which is consumed by A_2. A_2 outputs a pair (x', j) where x' is a guess at password number j. We call such an A_2 *challenge-respecting* if it only outputs (x', j) when it never made a query to its third oracle on input j.

Definition 3.1. *Let C be an oracle Turing machine which takes input $\vec{c}$ of length ks, maintains no long-term state, and outputs strings of length k. Let A_1 be an arbitrary $(q_{1,1}, q_{2,1}, q_3)$-BOTM and let A_2 be an arbitrary challenge-respecting $(q_1 - q_{1,1}, q_2 - q_{2,1}, n-1)$-BOTM. Define*

We say that C is $(q_1, q_2, q_3, \epsilon(k))$-password-securing if for all n and for all such A_1, A_2, $\mathsf{Adv}_{\mathsf{pwd}}(A_1, A_2, C, \mathcal{H}, k) < \epsilon(k)$.

This definition specifies that C produce a secure way of storing password-checking values when used with random salts. This use case was specifically envisioned by Spilman [4], although his construction of it is substantially simplified here.

Theorem 3.2. *The construction C of Section 2.1 is $(q_1, q_2, s2^{-e/k}, \epsilon(n))$-password-securing for $\epsilon(n) = \frac{nq_2}{2^k} + \frac{q_c^2}{2^k} + \frac{n2^e q_3^k}{s^k}$.*

Proof. This follows from Theorem 2.3. Suppose there is some adversary (A_1, A_2) and some k such that $\mathsf{Adv}_{\mathsf{pwd}}(A_1, A_2, C, \mathcal{H}, k) > \epsilon(k)$. Consider the following distinguisher D:

1. D runs A_1 on input k, making queries to its oracles in order to satisfy the queries A_1 makes. D receives output α.
2. D chooses $x_1, \ldots, x_k$ and $s_1, \ldots, s_k$ at random, and makes k queries to its first oracle to calculate $y_1, \ldots, y_k$.
3. D runs A_2 on input $\alpha, y_1, \ldots, y_k, s_1, \ldots, s_k$. For any queries A_2 makes to its first or second oracle, D answers that query by making a query to its first or second oracle. For queries A_2 makes to its third oracle, D returns the requested x_i. D receives output (x', j).
4. D makes a query to its first oracle on input $x'||s_j$ to calculate y'_j. If $y'_j = y_j$, D outputs 1, otherwise D outputs 0.

If D is run with the real oracles, it has probability $\frac{q_1+1}{2^e} + \epsilon(n)$ of outputting 1, by our assumptions.

If D is run with the simulator and a random $\mathcal{F}$ in place of $C^{\mathcal{H}}$, the probability of D outputting 1, we will prove, is at most $\frac{kq_2}{2^k} + \frac{q_c^2}{2^k} + \frac{k2^e q_3^k}{s^k} + \frac{q_1+1}{2^e}$ where $q_c = q_1 + k + 1 + \frac{q_2}{2+k(1-1/\log_2 s)}$.

The idea is to use the principle of deferred decisions to prove that, except for situations which occur with low probability, the probability that A_2 can output a correct password is $\frac{q_1+1}{2^e}$. We do this with an emulation of D that defers choosing the x_i values until it is necessary to do so:

1. We pick $\vec{c} \leftarrow \{0,1\}^{ks}$.
2. We run A_1 on input k, with the following methods for answering oracle queries:
   - For queries of A_1 to its first oracle, we run a random function oracle $\mathcal{F}$.
   - For queries of A_1 to its second oracle, we execute the simulator S of Theorem 2.3, which in turn also accesses the random function oracle $\mathcal{F}$.
   - For queries of A_1 to its third oracle, we return the requested block of $\vec{c}$.

We receive output α from A_1.

3. Choose $s_1, \ldots, s_n$ at random of length k. For each i:
   - Set z_i to be the number of queries to $\mathcal{F}$ of the form $x||s_i$ for some x of length e. With probability $z_i 2^{-e}$, choose one such input x at random and set $y_i = \mathcal{F}(x||s_i)$ and set $x_i = x$. Otherwise, pick y_i at random.
4. Run A_2 on input $\alpha, y_1, \ldots, y_k, s_1, \ldots, s_k$, with the following methods for answering oracle queries.
   - For unique queries of A_2 to its first oracle (or of the simulator) *not* of the form $x||s_i$ for x of length e and $i \in [1, k]$, query the input to $\mathcal{F}$.
   - For unique queries of A_2 to its first oracle (or of the simulator) of the form $x||s_i$ for x of length e: With probability $1/(2^e - z_i)$, set $x_i = x$ and set $\mathcal{F}(x||s_i) = y_i$. Otherwise, increment z_i and query $x||s_i$ to $\mathcal{F}$.
   - For queries of A_2 to its second oracle, we continue executing S, handling the queries S makes as above.
   - For a unique query of A_2 to its third oracle on input i, if x_i has been set, return x_i. Otherwise, pick a random value x of length e such that $\mathcal{F}(x||s_i)$ is not yet set, set x_i to be x, and set $\mathcal{F}(x||s_i) = y_i$.
5. Obtain the output (x', j) of A_2. If $\mathcal{F}(x'||s_j)$ has been set, return 1 if and only if $\mathcal{F}(x'||s_j) = y_j$. If $\mathcal{F}(x'||s_j)$ has not been set and x' is not of length e, pick $\mathcal{F}(x'||s_j)$ at random and return 1 if and only if $\mathcal{F}(x'||s_j) = y_j$.

   If x' is of length e, set $x_j = x'$ and return 1 with probability $1/(2^e - \alpha_j)$. Otherwise, pick $\mathcal{F}(x'||s_j)$ at random and return 1 if and only if $\mathcal{F}(x'||s_j) = y_j$.

The reader may verify that the behavior of this alternate process is exactly equivalent to a run of D with ideal oracles.

We identify an event bad that may occur in a run of this alternate setting. The event bad occurs if (1) some query of the form $(\mathbf{j}||x)$ is made by A_1 to its second oracle for x of length $k + e$, and for some $i \in [1, k]$, s_i is later chosen to be equal to the last k bits of x, (2) if two distinct random choices made by $\mathcal{F}$ result in the same random choice, or (3) if for any $1 \leq i \leq n$, there is some $x \in \{0,1\}^e$ such that for all k locations $\{\mathsf{loc}(S(\mathbf{j}||x||s_i))\}$ are locations revealed by queries by A_1 to its third oracle.

A_1 makes at most $q' \leq q_2$ queries to its second oracle so at most q_2 distinct bad choices for each s_i are possible. Therefore, bad occurs for this reason with probability at most $\frac{kq_2}{2^k}$.

As for the second type of bad event, there are three choices that we consider: adversary queries to the first oracle, simulator queries to the $\mathcal{F}$ oracle, and queries by D to the $\mathcal{F}$ oracle. The adversary makes at most q_1 queries to its first oracle total, and our emulation makes a total of $k+1$ random choices: the choice of the k y_i values, and checking the adversary's output. The simulator makes at most one query to $\mathcal{F}$ for every $k(1-1/\log_2 s)+2$ queries the adversary makes to its second oracle. Therefore, there are q_c total choices, so the probability of this second type of bad event is at most $q_c^2/2^k$.

The third type of bad event occurs with probability at most $k2^e(\frac{q_3}{s})^k$, since there are 2^e values in $\{0,1\}^e$ and k salts, and each such pair leads to a choice of k independent locations, which each have a probability of at most $\frac{q_3}{s}$ of being locations that are previously known.

The overall probability that bad occurs may thus be bounded by $\frac{kq_2+q_c{}^2}{2^k} + \frac{k2^e q_3^k}{s^k}$.

We now assume that bad does not occur, and prove a bound on the probability that the above alternative process outputs 1. Suppose the output of A_2 is (x',j). There are two cases.

First, if the value of x_j has been set, then we output 1 if $x' = x_j$ (which occurs with probability at most 1). The probability that x_j has been set without j being queried to A_2's third oracle is $\frac{\alpha_j}{2^e}$.

If the value of x_j has not been set, then we output 1 with probability $\frac{1}{2^e-\alpha_j}$, in other words, only if we set $x_j = x'$ during the check. (Other ways to output 1 would amount to a bad event of the second type.)

Now, $\frac{\alpha_j}{2^e} + (1-\frac{\alpha_j}{2^e})(\frac{1}{2^e-\alpha_j}) = \frac{\alpha_j+1}{2^e}$ bounds the probability that we output 1. This can in turn be bounded by $\frac{q_1+1}{2^e}$, the case in which all first oracle queries are made on e-bit passwords concatenated with s_j. $\square$

4 Discussion

In this paper, we present a construction of a Random Uncloneable Function based on Spilman's blind hash algorithm [4]. We prove two main results.

First, we prove that the construction is indifferentiable from random in the (adaptive) bounded retrieval model and is in this sense a random uncloneable function. Second, we prove that the password security scheme Spilman describes as an application of blind hashing is secure against passive adversaries in the bounded retrieval model.

Though passive security is generally weaker than active security, we are satisfied with this result; it shows that offline attacks are prevented, even when most of the corpus is known to the attacker. The adaptive security case (not discussed much in this paper) seems much more comparable to an *online* attack, which cannot really be prevented.

Both of these results are proven in an absolute sense, without any reliance on computational complexity assumptions such as the hardness of factoring or of computing discrete logarithms.

We close by noting that physically uncloneable functions have been shown to be useful in a variety of applications, and any such application is potentially an application of our construction. Unlike *physically* uncloneable functions, our random uncloneable function is not uncloneable in an absolute sense: honest parties may, through concerted effort, duplicate the entire corpus accurately and therefore achieve a kind of load balancing and redundancy that is impossible for physically uncloneable functions.

References

[1] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In *Annual International Cryptology Conference*, pages 430–448. Springer, 2005.

[2] Giovanni Di Crescenzo, Richard Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In *Theory of Cryptography Conference*, pages 225–244. Springer, 2006.

[3] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *Theory of Cryptography Conference*, pages 21–39. Springer, 2004.

[4] Jeremy Spilman. TapLink blind hashing technical specification, 2016. Technical whitepaper.