Error-Tolerant Password Recovery

Niklas Frykholm
RSA Laboratories
P.O. Box 10704
Stockholm, SE-121 29 Sweden
nfrykholm@rsasecurity.com

Ari Juels
RSA Laboratories
20 Crosby Drive
Bedford, MA 01730 USA
ajuels@rsasecurity.com

ABSTRACT

Many encryption systems require the user to memorize high entropy passwords or passphrases and reproduce them exactly. This is often a difficult task. We propose a more fault-tolerant scheme, where a high entropy key (or password) is derived from a sequence of low entropy passwords. The user is able to recover the correct key if she remembers a certain percentage of the passwords correctly. In contrast to other systems that have been proposed for fault-tolerant passwords, our basic design is provably secure against a computationally unbounded attacker.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption; E.4 [**Data**]: Coding and Information Theory—*Error control codes*; H.1.2 [**Models and Principles**]: User/Machine Systems—*Human factors*

General Terms

Human Factors, Security

Keywords

Error-correcting codes, fault-tolerance, fuzzy commitment, Reed-Solomon codes, password ensembles, password recovery

1. INTRODUCTION

Human memory is in a constant state of flux: Every day we memorize new facts and forget old ones. This is normal and unavoidable, but can sometimes have untoward consequences. If we forget a password that has been used to encrypt important data, the data may be lost. Forgotten passwords are in fact one of the most common problems confronting IT help-desks. This has prompted the creation of a number of different systems for *password recovery.* The aim of these systems is to provide reliable secondary means for legitimate users to recover lost passwords, without significantly increasing the vulnerability against attackers.

Traditional methods of password recovery rely on the use of trusted third parties. Web sites typically let a site administrator, a help-desk employee, or an automatic script e-mail the forgotten password to the user upon request (often in the clear). Alternatively, a lost password might be provided by telephone upon presentation of some identifying information, such as the user's Social Security Number. A more sophisticated, but less common technique is to use *secret sharing* [2, 11]. The password is divided into n shares in such a way that for reconstruction of the password it is necessary and sufficient to collect k of these shares. The n shares are distributed among entities trusted by the user.

All of these methods carry the disadvantage of reliance on third parties. In the case where passwords are furnished on request, the password must be held in explicit form by scripts or help-desk personnel. This heightens the potential for abuse of the password by insiders, and also makes the password vulnerable to outside attackers that can penetrate the outer defenses of the server. Such attacks are not uncommon, as illustrated by recent reports of passwords and credit card numbers being stolen from public Web sites [14]. Shamir's secret sharing provides better security guarantees, but has other shortcomings. The user must contact k trusted parties to reconstruct her password. Collecting the necessary shares thus requires getting in touch with a number of people who may be out of town or for other reasons unavailable. Alternatively, the user may store the shares on online servers, but this again heightens the potential for attack. All of these caveats apply equally well to systems for recovering strong cryptographic keys.

Ellison, Hall, Milbert, and Schneier [4] propose a recovery system based on what they refer to as *personal entropy.* In this system, the user is asked a number of questions about her personal history. The sequence of answers, which we denote by $\vec{w}$, is used to encrypt a randomly generated cryptographically strong key $\vec{c}$. That is, the key $\vec{c}$ is encrypted as a ciphertext $\vec{\delta} = \mathsf{Enc}_{\vec{w}}[\vec{c}]$ under a collection $\vec{w}$ of what may be thought of as ancillary passwords. If the user loses the key $\vec{c}$ (or, say, an important password encrypted under $\vec{c}$), she can recover it by furnishing the sequence $\vec{w}$ of ancillary passwords and performing a decryption operation Dec on $\vec{\delta}$. The critical property of the decryption scheme Dec is that it has error-correcting capabilities. By use of secret sharing techniques, the Ellison *et al.* system allows the user to recover the password even if some questions are answered incorrectly, i.e., even if $\vec{w}$ as furnished by the user is not entirely correct. The main advantage of this system is that it dispenses with the use of trusted third parties. The user

CCS'01, November 5-8, 2001, Philadelphia, Pennsylvania, USA.

can in principle safely store $\vec{\delta}$ on her laptop, or even in a public directory. Of course, the key $\vec{c}$ can be used to encrypt important passwords for the user in case of loss, or to encrypt other sensitive information or keys.

Since the questions in this system may be made to concern personal facts that the user is not likely to forget — e.g., the name of her first pet, her mother's maiden name, and so forth — the chance of a user recovering the password $\vec{c}$ is very good. Since no trusted third parties are involved, management of the password is easier, and vulnerability of the password to attack is diminished. If desired, traditional Shamir's secret sharing can be used in series with or parallel to the encryption, either strengthening the protection further or providing an alternative recovery mechanism of last resort. Thus the Ellison *et al.* scheme strikes an attractive balance between convenience and security.

A serious shortcoming of the Ellison *et al.* scheme, though, is its lack of rigorous security analysis. Normally, a question/answer system is as secure as the entropy of the answers. Ellison *et al.*, however, employ an *ad hoc* secret sharing scheme in their system that undermines conventional security guarantees. Bleichenbacher and Nyugen [3] demonstrated a serious weakness in the Ellison *et al.* system: With the parameters recommended for a security level of 2^{112}, the system is in fact vulnerable to an attack that requires only 2^{64} operations.

In this paper, we present a system similar in flavor to the one proposed by Ellison *et al.* The main difference is that we base our system on the *fuzzy commitment* technique developed by Juels and Wattenberg [9] rather than on more traditional secret sharing methods. This allows us to achieve a high degree of fault tolerance, but at the same time compute a rigorous upper bound on the attacker's chance of recovering the password, even in the presence of non-uniform password distributions. In fact, as we show, our system offers provable information-theoretic security, that is, security against a computationally unbounded attacker. Moreover, our system offers a higher level of error-tolerance than the original Ellison *et al.* system (if desired by the designer), while simultaneously providing stronger security. Our proposed techniques are fairly simple both from a conceptual standpoint and in terms of implementation requirements. We refer to our proposal as an *error-tolerant password recovery* scheme, denoted by ETPAR.

The scheme can be used by a site administrator who wants to make it possible for users to recover lost passwords, but avoid the dangers of storing them in the clear. It can also be employed by an individual user to create a secure backup of her personal passwords.

1.1 Organization

We present our basic goals and working notation in section 2. Details of our ETPAR scheme are given in section 3, followed by a detailed security analysis in section 4. In section 5, we enumerate and discuss the practical considerations faced by the system designer by discussing a basic prototype design. We offer some preliminary experimental results in section 6.

2. GOALS AND NOTATION

Our aim is to enable the user to recover a cryptographically strong secret key $\vec{c}$. We are most interested in scenarios in which this key $\vec{c}$ is used to encrypt an important password Π for recovery, but other uses are of course possible. In an initialization phase, that is, to prepare her password for recovery, the user chooses a collection of ancillary passwords, e.g., answers to personal questions. A secret which we denote by $\vec{w}$ is constructed from these ancillary passwords. The user then constructs a ciphertext $\vec{\delta} = \mathsf{Enc}_{\vec{w}}[\vec{c}]$, using an appropriate cipher Enc to encrypt $\vec{c}$ under $\vec{w}$. The user may store the ciphertext $\vec{\delta}$ in some easily accessible place or places, e.g., on a laptop, on a PDA, or in a public directory. To enable recovery of the password Π, the user also stores an encryption of Π under $\vec{c}$, i.e., $E_{\vec{c}}[\Pi]$ for some suitable symmetric cipher E.

Error-tolerant recovery of $\vec{c}$ (and thus the password Π) requires the user to decrypt $\vec{\delta}$. In particular, the user should be able to extract $\vec{c}$ from $\vec{\delta}$ given a collection of ancillary passwords yielding $\vec{w'}$ that is "close" to the secret $\vec{w}$. Thus we want a decryption scheme Dec such that $\mathsf{Dec}_{\vec{w'}}[\vec{\delta}] = \vec{c}$ iff $\vec{w'}$ is close to $\vec{w}$ in some appropriate metric. Another way of viewing this is that the user should be able to recover the key $\vec{c}$ if she can come up with enough of her original ancillary passwords. At the same time, we do not want an attacker with access to $\vec{\delta}$ to be able to recover $\vec{c}$. (In our system, the encryption function Enc will be simple addition over a field, i.e., we shall use $\vec{w}$ essentially like a random pad, while Dec will involve subtraction and Reed-Solomon decoding. Other cipher choices are possible, though.)

We consider scenarios in which the secret $\vec{w}$ is derived from a sequence of N passwords $[p_i]_{i=1}^{N}$ selected by the user. The passwords may be answers to personal questions, as explained above, or they may be obtained in some other way, e.g., as graphical passwords [8] or biometric information (see, e.g., [1]). We assume here that each password p_i is selected from a fixed distribution P_i over all bit strings, and that the password distributions $[P_i]_{i=1}^{N}$ are independent. We adopt this independence assumption for three reasons: First, because it is very difficult to estimate correlations among passwords in the real world; second, because we believe that passwords can and should be chosen so as to minimize such correlations; and third, because this assumption considerably simplifies our security analysis.

To minimize the non-uniformity in the distribution of $\vec{w}$, we fix a different mapping f_i for every password distribution P_i. This mapping $f_i : \{0,1\}^* \to \{0,\ldots,q-1\}^{l_i}$ takes a password p_i and converts it into a string w_i with l_i symbols from the alphabet $\mathcal{F} := \{0,\ldots,q-1\}$. Tailoring f_i and l_i carefully to compensate for non-uniformities in P_i is important to the security of our system, as we discuss in detail later in the paper. In the meantime, it is convenient to think of $[w_i]$ as a sequence of small, password-derived keys $w_1, w_2, \ldots, w_N$. We refer to these keys w_i as *passkeys* and to the sequence $[w_i]$ as a *passkey ensemble.*

Let $\vec{w} := w_1 \parallel w_2 \parallel \ldots \parallel w_N$ denote the passkey ensemble written as a symbol string. Let $n := \sum_{i=1}^{N} l_i$ be the length of this symbol string. We write W_i to denote the (fixed) distribution induced over the passkey w_i by the distribution P_i and the mapping f_i. Observe then that the probability distribution over $\vec{w}$, denoted by W, is fixed. This distribution W is equal to a joint distribution induced by independent distributions $[W_i]$. We let $\mathsf{pr}_i(w_i)$ denote the probability of w_i in the distribution W_i, i.e., the probability that the user selects w_i for the i'th part of her password ensemble. We denote by $\mathsf{pr}(\vec{w})$ the probability of a specific ensemble $\vec{w}$ in

W, i.e., the probability, over all possible ensembles, that a user selects $\vec{w}$ as her own. Since we assume passwords in an ensemble $\vec{w}$ are independent, $\mathsf{pr}(\vec{w}) = \prod_{i=1}^{N} \mathsf{pr}_i(w_i)$.

Our scheme requires the use of a (linear) q-ary error-correcting code over n-symbol strings. We let C denote our choice of such a code. The user's secret data will in our scheme be a codeword $\vec{c}$ drawn uniformly at random from C. (Of course, in practice, $\vec{c}$ can be used to encrypt a weaker password, data, or anything else desired by the user.)

The task of the attacker is to determine the secret value $\vec{c}$. In our security analysis, we consider the strongest possible adversary, namely an attacker that is computationally unbounded, i.e., has infinite computational resources at her disposal. This adversary is further assumed to have complete knowledge of the password distributions $[P_i]$ and the mappings $[f_i]$, and thus the distributions $[W_i]$ and W. Additionally, we assume that the attacker has access to all public data in the protocol, most notably the encryption and decryption functions Enc and Dec and the ciphertext $\vec{\delta}$. We aim to compute rigorous bounds on the probability of such an attacker determining $\vec{c}$.

3. OUR ETPAR SCHEME

We specify our ETPAR scheme in terms of two paired algorithms, Enc and Dec. Additionally, the scheme includes a sequence of mappings $[f_i : \{0,1\}^* \rightarrow \{0,\ldots,q-1\}^{l_i}]_{i=1}^{N}$, and a (n,k)-error-correcting code over the field $\mathcal{F} := \{0,\ldots,q-1\}$. We denote the set of codewords in this code by $C \subset \mathcal{F}^n$, and let $\mathsf{decode} : \mathcal{F}^n \rightarrow C$ denote a (polynomial-time) decoding algorithm.

The algorithm Enc takes as input a sequence of ancillary passwords $[p_i]$ selected by the user, generally in response to a fixed series of questions; it also takes as input a secret key (codeword) $\vec{c} \in C$, selected uniformly at random. Using the mapping sequence $[f_i]$, the algorithm Enc computes a string $\vec{w}$ representing the passkeys generated by the password sequence $[p_i]$. Finally, Enc combines this with $\vec{c}$ to yield a ciphertext $\vec{\delta}$.

To decrypt the ciphertext $\vec{\delta}$, the user furnishes her ancillary passwords again, possibly introducing some errors. The algorithm Dec takes the passkey sequence $\vec{w'}$ generated by this newly furnished password sequence and combines it with the ciphertext $\vec{\delta}$ to compute what may be thought of as a "perturbed" or corrupted codeword. The "perturbation" in this codeword corresponds to errors in the sequence of newly furnished passwords. In the final stage, the algorithm Dec uses a decoding procedure for the underlying error correcting code to remove these errors, yielding a codeword (secret key) $\vec{c'}$. The decryption succeeds, i.e., $\vec{c'} = \vec{c}$, if the number of errors is below the error correcting capability of the code.

More precisely described, our algorithms Enc and Dec are as follows:

Algorithm 1. Enc *($[p_1, p_2, \ldots, p_N]$, $\vec{c}$) $\rightarrow \vec{\delta}$*

1. *Each password p_i is mapped to a l_i-symbol representation $w_i = f_i(p_i)$.*
2. *The password ensemble is represented as*
$$\vec{w} = w_1 || \ldots || w_N.$$
3. *The ciphertext $\vec{\delta} = \vec{w} - \vec{c}$ is computed.*
4. *The ciphertext $\vec{\delta}$ is output.*

Algorithm 2. Dec *($[p'_1, p'_2, \ldots, p'_N]$, $\vec{\delta}$) $\rightarrow \vec{c'}$*

1. *Each password p'_i is mapped to a l_i-symbol representation $w'_i = f_i(p'_i)$.*
2. *The password ensemble is represented as*
$$\vec{w'} = w'_1 || \ldots || w'_N.$$
3. *A perturbed codeword $\vec{\gamma}$ is computed as $\vec{\gamma} = \vec{w'} - \vec{\delta}$.*
4. *A key $\vec{c'} = \mathsf{decode}(\vec{\gamma})$ is computed.*
5. *The key $\vec{c'}$ is output.*

3.1 Example Deployment

Let us consider a simple example of how an ETPAR scheme might be deployed. Alice wishes to encrypt her password Π to allow for recovery in case of loss. She buys a piece of password-recovery software for her laptop that includes an ETPAR scheme. This software prompts Alice for her password Π, and then asks a series of N questions, such as "*What was the name of your first pet?*" or "*What did you give your mother for her 50th birthday?*". Alice replies with answers $p_1, p_2, \ldots p_N$.

Given Alice's answers, the password-recovery software selects a secret key $\vec{c} \in C$ uniformly at random ($|C|$ should be large enough to prevent brute force attacks). The software computes ciphertext

$$\vec{\delta} \leftarrow \mathsf{Enc}([p_1, p_2, \ldots, p_N], \vec{c})$$

and a second ciphertext $e = E_{K(\vec{c})}[\Pi]$, where E is some suitable symmetric cipher, such as AES and K is a key generation function seeded with $\vec{c}$. The pair of ciphertexts $(\vec{\delta}, e)$ are placed in a file on Alice's laptop.

Suppose now that Alice loses her password Π. She invokes the password-recovery software on the file containing the pair $(\vec{\delta}, e)$. The software now poses the original set of questions, to which Alice responds with answers $p'_1, p'_2, \ldots, p'_N$. The software computes

$$\vec{c'} \leftarrow \mathsf{Dec}([p'_1, p'_2, \ldots, p'_N], \vec{\delta})$$

and then $\Pi' = D_{K(\vec{c'})}[e]$, where D is the decryption algorithm corresponding to E. Provided that the answers Alice provides for the decryption are similar to the answers she provided for encryption, she will obtain her original password $\Pi' = \Pi$. Otherwise, the recovery will fail, i.e., we will (with overwhelming probability) have $\Pi' \neq \Pi$.

4. SECURITY ANALYSIS

We wish to prove that a computationally unbounded attacker with access to the encrypted value $\vec{\delta}$ and knowledge of the password distribution has only a small chance of guessing the secret $\vec{c}$. Note that guessing $\vec{c}$ is equivalent to guessing $\vec{w}$, since $\vec{c} = \vec{w} - \vec{\delta}$.

We let $p_A(\vec{\delta})$ denote the probability that the best possible attacker is able to guess $\vec{w}$ when the encrypted value is $\vec{\delta}$. Two measures of interest are then p_A^{avg}, the average of $p_A(\vec{\delta})$ over all values of $\vec{\delta}$ and p_A^{worst}, the maximum value for a particular $\vec{\delta}$.

The quantity p_A^{avg} is the attacker's chance of breaking into a particular account where the user has picked $\vec{w}$ according to the distribution W. The quantity p_A^{worst} is the attacker's chance of breaking into an account when she gets to pick the value of $\vec{\delta}$. In practice, this can happen if a large number of users have their $\vec{\delta}$ stored in public directories. The attacker can then target the user with the most vulnerable value of $\vec{\delta}$. If the attacker only knows $\vec{\delta}$ for a small number of accounts, her probability of success will be closer to p_A^{avg} than to p_A^{worst}.

A system is secure against both these attacks if we can prove that both p_A^{avg} and p_A^{worst} are small (of course $p_A^{worst} \geq p_A^{avg}$).

4.1 Bounding p_A^{avg}

Since C is a linear code and $\vec{c}$ is picked uniformly at random, $\vec{\delta} = \vec{c} - \vec{w}$ will be a random vector from the coset of C that contains $\vec{w}$.[1] The only information that $\vec{\delta}$ gives the attacker is thus which coset $\vec{w}$ belongs to.

Let S^* denote the cosets of C and $S_x \in S^*$ the coset that x belongs to. Given a particular value of $\vec{\delta}$, the best possible attacker guesses the password $\vec{w}$ which is most probable in the coset $S_{\vec{\delta}}$. This attacker's probability of success is

$$p_A(\vec{\delta}) = \max_{\vec{w} \in S_{\vec{\delta}}} \frac{\mathsf{pr}(\vec{w})}{\mathsf{pr}_{S^*}(S_{\vec{\delta}})}. \qquad (1)$$

Here, $\mathsf{pr}_{S^*}(S_{\vec{\delta}})$ denotes the total weight on the coset $S_{\vec{\delta}}$. That is, $\mathsf{pr}_{S^*}(S_{\vec{\delta}})$ may be viewed as the probability that a passkey ensemble drawn from the distribution W belongs to the coset $S_{\vec{\delta}}$, i.e., $\mathsf{pr}_{S^*}(S_{\vec{\delta}}) := \sum_{\vec{w} \in S_{\vec{\delta}}} \mathsf{pr}(\vec{w})$.

We write $p_A(S_{\vec{\delta}}) := p_A(\vec{\delta})$, since the probability is the same for all elements of the coset.

Averaging (1) over S^* we get:

$$p_A^{avg} = \sum_{S \in S^*} \mathsf{pr}_{S^*}(S) p_A(S) = \sum_{S \in S^*} \max_{\vec{w} \in S} \mathsf{pr}(\vec{w}). \qquad (2)$$

So p_A^{avg} is the sum of the probabilities of the most probable item in each coset of C.

A linear (n, k)-code has q^{n-k} cosets. This means we can bound (2) as

$$p_A^{avg} \leq q^{n-k} \max_{\vec{w} \in \mathcal{F}^n} \mathsf{pr}(\vec{w}).$$

It is more convenient to work with logarithmic expressions, so we define $\Phi_{avg} := -\log p_A^{avg}$. The quantity Φ_{avg} is a logarithmic measure of the attacker's average chance of guessing a password. It thus corresponds to the entropy of a conventional password system. An ETPAR system with $\Phi_{avg} = 60$ will be as secure as an ordinary password system with 60 bits of entropy. We get:

$$\Phi_{avg} \geq (k - n) \log q + H_{min}(\vec{w}), \qquad (3)$$

where $H_{min}(\vec{w})$ is the *minimum entropy* or *minentropy* of $\vec{w}$, defined in the standard way as

$$H_{min}(\vec{w}) := -\log \max_{\vec{w}} \mathsf{pr}(\vec{w}).$$

Recalling that $\vec{w}$ is a symbol string of length n, we see that the security of the system is thus determined by the entropy of the codewords $k \log q$ and the non-uniformity/redundancy in $\vec{w}$, expressed as $n \log q - H_{min}(\vec{w})$. In [6], a similar expression is derived using a different argument.

The purpose of the $f_i : p_i \mapsto w_i$ mapping used in the protocol is to reduce this redundancy as much as possible.

We can break (3) into N parts to see the effect that each password in the ensemble has on the security. It seems reasonable to attribute to each passkey its entropy $H_{min}(w_i)$ and a share of $(k - n) \log q$ proportional to l_i. To this end we define:

$$\Phi_{avg,(i)} := l_i \left(\frac{k}{n} - 1 \right) \log q + H_{min}(w_i).$$

Summing this gives the RHS of (3), so:

$$\Phi_{avg} \geq \sum_{i=1}^{N} \Phi_{avg,(i)}.$$

Recall that the mappings $[f_i]$ shape the passkey distributions $[W_i]$. Thus, it is important to select these mappings so as to maximize Φ_{avg}. We discuss ways of achieving this in section 4.3.

4.2 Bounding p_A^{worst}

Restating (1) as

$$p_A(\vec{\delta}) = \max_{\vec{w} \in S_{\vec{\delta}}} \frac{\mathsf{pr}(\vec{w})}{\sum_{\vec{w'} \in S_{\vec{\delta}}} \mathsf{pr}(\vec{w'})}, \qquad (4)$$

it is easy to see that when $\vec{w}$ is distributed uniformly over all n-symbol strings, we have $p_A(\vec{\delta}) = 1/|S_{\vec{\delta}}| = 1/|C|$. In other words, the attack probability is exactly the probability of guessing a random codeword $\vec{c}$.

In general, however, we cannot assume that the passkey distributions $[W_i]$ will be uniform. We characterize the user's choice of passkey w_i in terms of two measures:

$$min_i := \min_{w_i \in \mathcal{F}^{l_i}} \mathsf{pr}_i(w_i)$$

$$max_i := \max_{w_i \in \mathcal{F}^{l_i}} \mathsf{pr}_i(w_i).$$

By our independence assumption on passwords and thus passkeys, the numerator in (4) is bounded above by

$$\prod_{i=1}^{N} max_i.$$

As for the denominator, we have

$$\sum_{\vec{w} \in S_{\vec{\delta}}} \mathsf{pr}(\vec{w}) \geq |S_{\vec{\delta}}| \left(\min_{\vec{w} \in \mathcal{F}^n} \mathsf{pr}(\vec{w}) \right) = |C| \prod_{i=1}^{N} min_i.$$

Thus we have[2]

[1] Two vectors $\vec{a}$ and $\vec{b}$ belong to the same coset of C iff $\vec{a} - \vec{b} \in C$. Note that each vector in $\mathcal{F}^n$ belongs to exactly one coset of C.

[2] We can also obtain the stricter but more complex expression

$$p_A^{worst} \leq \frac{1}{1 + |C - 1| \prod_{i=1}^{N} \frac{min_i}{max_i}}.$$

$$p_A^{worst} \leq |C|^{-1} \prod_{i=1}^{N} \frac{max_i}{min_i}.$$

This result is not completely satisfactory. Note most importantly that we cannot bound the probability if $min_i = 0$ for some i. Thus, if the attacker knows a single password in the sequence $[p_i]$, she might be able to guess the entire codeword.

The reason why we cannot bound p_A^{worst} in this situation is that the distribution of the codewords might coincide with the distribution of the passwords in such a way that a particular coset $S_{\vec{\delta}}$ only contains a single possible password.

For example, suppose that the first passkey w_1 is three symbols long and that the attacker knows that $w_1 =$ '000'. Suppose also that only one codeword starts with the pattern '111' and that $\vec{\delta}$ also starts with '111'. Then, the attacker will know with certainty that the secret is the one codeword that starts with '111'.

To get a better bound for p_A^{worst} we must place some restrictions on the code C.

Definition 1. A q-ary code C is m-wise uniform if for any sequence of m distinct indices $[i_1, i_2, \ldots, i_m]$ and any sequence of values $[x_1, x_2, \ldots, x_m]$ picked from the q code symbols:

$$|\{c \,|\, c \in C, c_{i_1} = x_1, c_{i_2} = x_2, ..., c_{i_m} = x_m\}| = \frac{|C|}{q^m}.$$

In other words, the code is m-wise uniform if every pattern of m symbols occurs with equal frequency in the code. It is easy to see that if C is m-wise uniform, its cosets $S_{\vec{\delta}}$ are also m-wise uniform. Two classes of codes with good uniformity are the (n, k)-MDS codes, which have uniformity k, and the self-dual codes with distance d, which have uniformity $d-1$. This is proved in appendix A.

The notion of m-wise uniformity can be used to compute a better security bound. We partition the full set of passkey indices $I = \{1, 2, ..., N\}$ into two sets I_1 and I_2, where $l_{I_2} := \sum_{i \in I_2} l_i \leq m$. This lets us bound the denominator in (4) as:

$$\sum_{\vec{w} \in S_{\vec{\delta}}} \mathsf{pr}(\vec{w}) = \sum_{\vec{w} \in S_{\vec{\delta}}} \left(\prod_{i \in I_1} \mathsf{pr}_i(w_i) \prod_{i \in I_2} \mathsf{pr}_i(w_i) \right) \geq$$

$$\geq \left(\prod_{i \in I_1} min_i \right) \sum_{\vec{w} \in S_{\vec{\delta}}} \prod_{i \in I_2} \mathsf{pr}_i(w_i) = \frac{|C|}{q^{l_{I_2}}} \prod_{i \in I_1} min_i,$$

where the last step makes use of the fact that the sum runs over all possible symbol patterns, since $S_{\vec{\delta}}$ is m-wise uniform and $l_{I_2} \leq m$.

We thus obtain:

$$p_A^{worst} \leq |C|^{-1} \prod_{i \in I_1} \frac{max_i}{min_i} \prod_{i \in I_2} q^{l_i} max_i.$$

As before, we define $\Phi_{worst} := -\log p_A^{worst}$. Since Φ_{worst} measures the difficulty of guessing the worst password, it corresponds to the *minentropy* of an ordinary password system. An ETPAR system with $\Phi_{worst} = 60$ corresponds to an ordinary password system with a minentropy of 60 bits. We obtain:

$$\Phi_{worst} \geq k \log q - \sum_{i \in I_1} \log \frac{max_i}{min_i} - \sum_{i \in I_2} (\log max_i + l_i \log q).$$

As before, we want to see the contribution of each passkey, so we define:

$$\Phi_{worst,(i)} := \frac{kl_i}{n} \log q - \log max_i + \log min_i. \qquad (5)$$

Now,

$$\Phi_{worst} \geq \sum_{i \in I_1} \Phi_{worst,(i)} + \sum_{i \in I_2} \Phi_{avg,(i)}.$$

The uniformity of the code lets us use $\Phi_{avg,(i)}$ instead of $\Phi_{worst,(i)}$ for troublesome passwords. We can select I_2 freely under the constraint $l_{I_2} \leq m$. If we can move all passkeys w_i such that $min_i = 0$ into I_2, we are able to compute a security bound where this would otherwise not be possible. When $min_i \neq 0$, moving w_i from I_1 to I_2 gives us $\Phi_{avg,(i)} - \Phi_{worst,(i)} = -\log min_i - l_i \log q$ more bits of security. We get the best security bound if we move into I_2 those passkeys w_i that have the highest value of this last expression.

4.3 The Effects of Mapping

The quantities p_A^{avg} and p_A^{worst} depend not only on the distribution of passwords and codewords, but also on the way the mapping from passwords $[p_i]$ to passkeys $[w_i]$ is performed by the mapping functions $[f_i]$. We want to maximize (5) by having a large value for $l_i \log q$ while making sure that the distribution is as uniform as possible ($max_i \approx min_i$).

The mapping function $f_i : p_i \mapsto w_i$ for the ith password acts as a bucket sort, mapping each password in the input space to one of the q^{l_i} buckets in the output space. Thus, f_i will in fact be a hash function, but in the conventional rather than the cryptographical sense (we do not depend on f_i being hard to invert). To avoid confusion, we will continue to refer to f_i as a *mapping* or a *bucket sort*.

If we have complete knowledge of a password distribution P_i, then we can construct a customized mapping f_i that distributes the passwords evenly among the buckets. We then get $max_i \approx min_i$ and $\Phi_{worst,(i)} \approx kl_i \log q/n$. Of course, an even distribution is only possible if $q^{-l_i} \geq p_i^{max}$, where p_i^{max} is the probability of the most probable password. We can write this as $l_i \log q \leq H_{min}(P_i)$, so the security is bounded by the minentropy of the passwords, as should be expected.

Since the range of the mapping functions is $\mathcal{F}^{l_i}$ we cannot select the number of buckets freely, it must be a power of q. This restriction means that entropy will sometimes have to be sacrificed (we must use fewer buckets than we would like to). An alternative is to allow an arbitrary output range for f_i and use a more complex transformation from $[w_i]$ to $\vec{w} \in \mathcal{F}^n$. The disadvantage of this approach is that it makes the equations more complex. Also, since passkeys and code symbols will no longer coincide, erroneous passkeys will corrupt more symbols.

If we have no knowledge of the distribution, the best we can do is to use a random mapping. Standard hash functions such as MD5 and SHA-1 will essentially act as random mappings. The problem with using a random mapping is that it will typically distribute the passwords unevenly, with some buckets containing more passwords than others ($max_i > min_i$). A detailed analysis of the effects of using a random mapping is found in appendix B. For a typical case, a random mapping can reduce the entropy by as much as 85 % from what is achievable with a customized mapping.[3]

The conclusion is that to get good security we should use a mapping that has been tailored as closely as possible to the password distribution. Section 5.2 discusses how this can be achieved in a practical setting.

4.4 Computationally Bounded Attackers

So far we have only considered the security against a computationally unbounded attacker. We have assumed that the attacker will always be able to find the most probable password in $S_{\vec{\delta}}$ and that she will be able to distinguish "weak" $\vec{\delta}$ from "strong" $\vec{\delta}$.

For a polynomially bounded attacker neither task is easy. Enumerating all passwords in $S_{\vec{\delta}}$ requires q^k operations and is thus on par with guessing the secret by brute force. So the attacker must try a more sophisticated approach. But today, there are no known polynomial time algorithms that accomplish this task for Reed-Solomon codes.

The offset together with the password distribution gives a probability distribution for the symbols in the codeword. The task of the attacker is to compute a list of the most probable codewords given this symbol distribution. The problem is related to the problem of *decoding from uncertain receptions* where η options are given for each codeword symbol and we want to find all codewords matching this pattern. Guruswami and Sudan [7] show that this problem can be solved in polynomial time for an (n, k)-Reed-Solomon code, provided that $\eta < n/k = 1/R$. This is the best known result for this problem. We will typically use a code rate of $k/n = 1/2$ which means that the Guruswami-Sudan method cannot be used unless $\eta = 1$, in which case decoding is trivial anyway.

This result illustrates the large gap between the capabilities of the unbounded and the bounded attacker using the current state of the art. The unbounded attacker can, in our model, compute the codeword provided that she can rule out one option for $m + 1$ of the symbols in the codeword, where m is the uniformity of the code. With state-of-the-art techniques, it is asymptotically infeasible for the bounded attacker to compute the codeword efficiently (in polynomial time) even if she can rule out *all but two* options for *each* symbol in the code. (Of course, practically speaking, with small enough parameters brute force search of the keyspace might be within reach.)

The weaknesses induced by random mapping thus seem to be much less serious when we consider computationally bounded attackers. Still, there are no hardness proofs for these problems and coding theory is a rapidly advancing field. If and when more is known about the computational complexity it might be possible to relax the requirements and use random mapping functions while still achieving good security guarantees.

5. PROTOTYPE SYSTEM DESIGN

To evaluate the proposals in this paper in a practical setting and provide reference parameters, we developed a prototype system. In this prototype, a local text file t is encrypted under a key $K(\vec{c})$ derived from the random codeword $\vec{c}$ by a key derivation function K. The purpose of K is to convert $\vec{c}$ into a format compatible with the choice of system cipher and to act as a time-lock that slows down brute-force attacks. The encrypted file $E_{K(\vec{c})}(t)$ is stored on the user's hard drive, together with the ETPAR encryption of the codeword, $\vec{\delta} = \mathsf{Enc}([p_1, ..., p_N], \vec{c})$. When the user wants to read the file, her password ensemble is used to decrypt $\vec{c}$, which in turn is used to decrypt the text file. In this section of the paper, we discuss the design choices made in the construction of the prototype.

The current prototype runs on NT and Windows 98 systems and requires very little in terms of system resources. Since it uses the platform-independent wxWindows library for all GUI operations we expect that it can be easily ported to other systems.

5.1 Security Level

The attack model we consider is one in which, having stolen the hard drive of the user, an attacker wants to access the encrypted data. Since the attacker will be able to guess the password offline we aim for a minimum of roughly $\Phi_{worst} \approx 60$ to achieve acceptable security, i.e., a 60-bit security level.[4]

In a situation in which an online throttling mechanism affords extra protection, it might be sufficient to have a substantially lower security level, e.g., $\Phi_{worst} \approx 14$, a 14-bit security level. This might be the case, for example, where the password protected by the ETPAR system is used in conjunction with the *server-assisted password hardening* protocol by Kaliski and Ford [5]. This system uses multiple authentication servers to ensure effective throttling even if one server database is compromised.

5.2 Questions & Hashing

The password ensemble in the prototype is formed by the user's answers to a set of questions about her personal history. Careful consideration is needed in the construction of such questions. We want as much entropy as possible in each question, but we also want to make sure that the user is not likely to forget the answer and that she will be able to produce consistent input. Answers to questions about the user's childhood or youth are usually more stable than questions about opinions or the present (e.g. compare "*What was your first job?*" with "*What is your current job?*" — the first question is both more stable and harder to guess).

Answers that are semantically but not syntactically equivalent pose another problem. If a question is not carefully phrased, it might be possible for a user to answer it in sev-

[3] In view of the leftover hash lemma [13], it might at first glance seem that random hashing could *increase* the uniformity, but that assumes that the mapping is unknown to the attacker. In our case the attacker has access to all public information and knows which mapping was used.

[4] Some may argue that we need even more, but since password recovery protocols are run so infrequently we can afford to use a key derivation function K that takes a full CPU minute or more. An example of a key derivation function can be found in [10]. The SETI@home project, which has been called the largest computing effort in human history, has currently used about 2^{38} CPU minutes. Breaking a 2^{60} system with one minute key derivation by brute force would thus require an effort corresponding to 2^{22} SETI@home projects. This seems to be a reasonable safety margin.

eral different ways, e.g. "*car*", "*the car*", or "*my car*". This must be taken into account when the question is written. It sometimes help to rewrite the question as a sentence with a fill-in answer, e.g. "*My favorite toy was a ...*".

Some syntactical differences can be fixed by *normalizing* the input. For example, our prototype strips punctuation, whitespace and case from the input before deriving the key, so "*John A. Doe*" and "*john a doe*" are viewed as equivalent. The idea of normalization might be taken even further — although we do not do so in our prototype. For example, common words such as "*the*", "*and*", and "*of*" could be stripped and the words in the input could be sorted alphabetically. That way "*Matt and Sue*" and "*sue, matt*" would be equivalent input. It should be noted that if normalization is taken too far, entropy loss will occur.

Some of the questions used in the prototype are: "*What was the name of the first boy/girl you kissed?*", "*Where did you celebrate the millennium?*", "*What was the name of your first pet?*".

Because of the many delicate issues surrounding question design: stability over time, entropy estimation, mapping, normalization, etc., we do not believe that users should be trusted with designing their own questions. Still, some flexibility is desired. For example, the third question above will be nonsensical to users who have never owned a pet. We can achieve this by letting the user select a set of questions from a library designed by security and human interface experts. For identification, only the selected questions are asked. This functionality is implemented in the prototype.

To ensure that the system meets the security demands, we must estimate the entropy of each question and tailor a mapping function f_i to the distribution of the answers. To do this properly, we must know the distribution of the answers (using a random mapping is, as we have seen, not advisable and overestimating the entropy will reduce the security of the system). For some questions, national census data or other wide scale investigations can give information about the distribution, e.g., Statistics Sweden [12] has complete information about the distribution of Swedish first names and surnames. When no public data are available, a survey can be used to determine the distribution at a moderate cost. An on-line site using the ETPAR system could gather such statistics as part of its normal operation.

For evaluation of our prototype, we pretend that such a survey has been conducted, i.e., when estimating the security of the system, we assume that we have access to mapping functions f_i that perfectly match the distributions of the answers.

5.3 Code Selection

The prototype uses Reed-Solomon codes, since they have excellent distance and uniformity. A Reed-Solomon code is a q-ary (n, k)-MDS code with, $q = t^s$, $n \leq t^s - 1$ and $k \leq n$. It has distance $d = n - k + 1$, uniformity k and can correct $\lfloor (n-k)/2 \rfloor$ errors or $n - k$ erasures (an *erasure* occurs when the value of a certain code symbol is unknown, i.e., if a question is left unanswered). We will only use codes with $t = 2$.

The code size s is determined by two factors: how large n needs to be (the number of passwords in the system) and the entropy of the passwords. Note that if s is selected too large, redundancy will be introduced in the passkeys. For example, if $q = 2^7$ and the passwords have 4 bits of entropy, we will have at least 3 bits of redundancy in the passkeys, regardless of which mapping we use. Preferably, s should be close to the entropy of the passwords (with $l_i = 1$) or the entropy should be a multiple of s (with $l_i > 1$).

If we are forced to use passwords with an entropy smaller than s one option is to concatenate several of these small passwords before calculating the passkey. When the total entropy is larger than s we can get rid of the redundancy. But note that an error in a single one of these smaller passwords will spoil the entire passkey.

The rate of the code, $R := k/n$, determines the amount of error correction and the number of passwords needed to attain a certain security level. With a code rate R we can correct approximately a $((1 - R)/2)$-fraction of errors or a $(1 - R)$-fraction of erasures. The number of passwords we need is proportional to $1/R$. Values in the vicinity of $R = 1/2$ often strike a good balance between fault-tolerance and brevity.

In the prototype each answer is mapped to an 8-bit representation. It is therefore suitable to use Reed-Solomon codes over $GF(2^8)$. We assume that the mapping is perfect, so that each passkey holds 8 bits of entropy. Each passkey then contributes $8R$ bits of security to the system and the total security with n passkeys is therefore $8Rn = 8k$.

The prototype uses $k = 7$, which gives us about 56 bits of security. (With one minute key derivation this would correspond to 2^{18} SETI projects; to allow rapid experimentation the prototype uses much faster key derivation.) To get a suitable code rate we set $n = 15$. This gives us a system with 15 questions where the user is allowed to answer 4 questions incorrectly or give 8 blank answer (or, more generally, to make τ errors and $8 - 2\tau$ omissions).

6. PRELIMINARY EXPERIMENT

In a preliminary experiment we tested the prototype on nine colleagues. They first selected their passwords in response to the questions posed by the prototype (with some flexibility in selection of questions). The experimental subjects subsequently refrained from using the program for one week, and then attempted to access the data by answering the questions again. (A longer period of time would have been desirable, but was impractical for this preliminary experiment.) Of the nine subjects, all were able to correctly recover their data, but one subject required more than one attempt.

On average the subjects made 1.6 errors, well below the correction limit. The errors were somewhat unevenly distributed. Two of the subjects made three errors and one subject made four errors.

57 % of the errors made by the subjects were simple syntactic errors that could easily have been handled by a better normalizing process, if there had been time to iterate the prototype design. For example, when asked about their doctor's name, some people alternated between answering "*X*" and "*Dr. X*". When asked about a date, the date format varied. When asked about a street name the formulations "*X*" and "*X St.*" were used.

If the normalization process had been modified to handle these simple cases, the error rate would have dropped from 1.6 to 0.5, one user would have two errors and the remaining users zero or one. We would thus be far below the correction limit. Some further statistics:

- 29 % of the errors were harder syntactical errors, for example writing "*Jen*" instead of "*Jennifer*" or "*carpet*" instead of "*Persian carpet*". It might be possible to eliminate some of these errors by rewriting the questions to disambiguate the answers, but some errors will probably always remain in this category.
- 7 % of the errors were semantic errors, where the user gave completely different answers.
- 7 % of the errors were spelling mistakes. We believe that in many cases, the user is likely to correct such mistakes herself on her next login attempt.

In conclusion, the ETPAR system seems to offer very reliable data recovery.

7. ACKNOWLEDGMENTS

The authors wish to extend their thanks to Madhu Sudan, Burt Kaliski, Håkan Andersson and the anonymous reviewers of this paper for their comments and suggestions.

8. REFERENCES

[1] Biometrics consortium, 2001. Website at http://www.biometrics.org.

[2] G. Blakely. Safeguarding cryptographic keys. In *AFIPS Conference Proceedings 1979: National Computer Conference*, volume 48, pages 313–317, June 1979.

[3] D. Bleichenbacher and P. Q. Nguyen. Noisy polynomial interpolation and noisy Chinese remaindering. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT '00*, pages 53–69. Springer-Verlag, 2000. LNCS no. 1807.

[4] C. Ellison, C. Hall, R. Milbert, and B. Schneier. Protecting secret keys with personal entropy. *Journal of Future Generation Computer Systems*, 16(4):311–318, Feb. 2000.

[5] W. Ford and B. S. K. Jr. Server-assisted generation of a strong secret from a password. In *Proceedings of the IEEE 9th International Workshop on Enabling Technologies (WETICE)*, Gaithersburg MD, June 2000. NIST.

[6] N. Frykholm. Passwords: Beyond the terminal interaction model. Master's thesis, Umeå University, Department of Computing Science, 2000. UMNAD 298/2000.

[7] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE TOIT: IEEE Transactions on Information Theory*, 45:1757–1767, Oct. 1999.

[8] I. Jermyn, A. Mayer, F. Monrose, M. K. Reiter, and A. D. Rubin. The design and analysis of graphical passwords. In *8th USENIX Security Symposium*, pages 1–14, Washington, D.C., USA, Aug. 1999. USENIX.

[9] A. Juels and M. Wattenberg. A fuzzy commitment scheme. In *5th ACM Conference on Computer and Communications Security*, pages 28–36, Singapore, Nov. 1999. ACM Press.

[10] RSA. PKCS #5: Password-based cryptography standard 2.0, Mar. 1999. http://www.rsasecurity.com/rsalabs/pkcs/pkcs-5/index.html.

[11] A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, Nov. 1979.

[12] Statistics Sweden, 2001. Website at http://www.scb.se/eng/index.asp.

[13] D. Stinson. Universal hash families and the leftover hash lemma, and applications to cryptography and computing. http://citeseer.nj.nec.com/stinson01universal.html.

[14] Hacker takes credit-card numbers. *Washington Post*, (Tuesday, January 11):E02, 2000.

APPENDIX

A. SECURITY DETAILS

The theorem below shows how the uniformity of any linear code C can be calculated.

THEOREM 1. *If C is a linear q-ary (n, k)-code with generator matrix G, then C is m-wise uniform if and only if every set of m columns from G is linearly independent.*

PROOF. The codeword corresponding to a message α is computed as $c = \alpha G$. Let $I = [i_1, \ldots, i_m]$ be a sequence of distinct integers and $x = [x_1, \ldots, x_m]$ be a sequence of code symbols. The number of codewords matching this pattern is then the number of solutions to the equation $x = \alpha G'$, where G' consists of columns $i_1, \ldots, i_m$ from G.

If every set of m columns from G is linearly independent, then G' will have rank m for every value of I. Since α has k degrees of freedom, the equation will have q^{k-m} solutions for every value of x. The code is thus m-wise uniform.

If every set of m columns from G is not linearly independent, then we can find a sequence I such that $\mathsf{rank}(G') \leq m - 1$. Since x has m degrees of freedom, there must exist an x such that the equation has no solutions, thus the code is not m-wise uniform.

□

COROLLARY 1. *If C is a linear (n, k)-code, C is m-wise uniform if and only if its dual code has distance at least $m + 1$.*

COROLLARY 2. *If C is a self-dual linear (n, k, d)-code, then C is $(d - 1)$-wise uniform.*

It is evident from the definition that no linear (n, k)-code can be more than k-wise uniform. The theorem below shows when this bound is achieved.

THEOREM 2. *If C is a linear $q - ary$ (n, k, d)-code, then C is k-wise uniform if and only if C is maximum distance separable ($d = n - k + 1$).*

PROOF. Assume that C is maximum distance separable. Since C is linear, the smallest Hamming weight of any nonzero codeword in C is $d = n - k + 1$.

Suppose c and c' are two codewords whose values agree in k positions. By the linearity of the code $c'' = c - c'$ is then a codeword with Hamming weight at most $n - k$. By the previous result, the only such codeword is $c'' = 0$ which means that $c = c'$.

Thus, there can exists at most one codeword that matches a specific k-symbol pattern. Since there are q^k such patterns and q^k codewords, each pattern must occur exactly once in the code. Thus, C is k-wise uniform.

Assume C is k-wise uniform. Then there exists exactly one codeword, the zero codeword, which matches a pattern consisting of k zeros. Thus, the non-zero codeword with smallest Hamming weight cannot have more than $k-1$ zeros, so $d \geq n-k+1$, which means that C is maximum distance separable.

□

B. THE EFFECTS OF MAPPING

To see how the mapping affects the security, let f be a bucket sort that maps A passwords into B buckets. Let $a_1, \ldots, a_A$ be the passwords and $b_1, \ldots b_B$ be the buckets. Let α_i be the probability that the password is a_i and β_i the probability that it is in bucket b_i.

The security we get with this mapping, based on (5), will then be:

$$\Phi = R \log B - \log \beta_{max} + \log \beta_{min}.$$

where $\beta_{min} := \min_{i \in [1,B]} \beta_i$ and $\beta_{max} := \max_{i \in [1,B]} \beta_i$. We should select f_i so that this expression is maximized. (For the moment, we ignore the fact that $B = q^{l_i}$ in our system and allow B to take on any integer value.)

If the system designer has complete knowledge of the password distribution α_i, she can easily select f so that Φ is maximized. To estimate the security in this case, let α_{max} be the probability of the most probable password and assume that the probabilities for the remaining $A-1$ passwords can always be distributed evenly among the remaining buckets. We will then have

$$\Phi = R \log B \qquad \text{for } B < \frac{1}{\alpha_{max}}$$

$$\Phi = R \log B - \log \alpha_{max} + \log \frac{1-\alpha_{max}}{B-1} \qquad \text{for } B \geq \frac{1}{\alpha_{max}}$$

This expression is maximized by $B = 1/\alpha_{max}$, which gives us $\Phi = -R \log \alpha_{max} = RH_{min}(\alpha)$. In this case the security depends only on the initial password minentropy and the amount of error correction (the R parameter).

Table 1: Security of random mapping

A	B	R	β_{max}	β_{min}	Φ
10	2	0.5	6.0	3.9	-0.12
100	2	0.5	53	46.6	0.30
100	5	0.5	25	15	0.48
100	10	0.5	15	5.7	0.27
100	5	0.2	25	15	-0.20
100	5	0.8	25	15	1.2

With a random mapping, such as MD5, the passwords will be more unevenly distributed. To simplify the analysis, assume that all A passwords have equal probability $\alpha = \frac{1}{A}$. β_{max} and β_{min} are then determined only by how many passwords end up in each bucket. This will be binomially distributed with $p = 1/B$.

The average value of Φ in this case will always be infinitely negative, since there is a finite probability that $\beta_{min} = 0$. This does not mean that the system will be insecure, because the m-wise uniformity lets us handle some zero min_i values. Rather, it means that the average value of Φ is not a very good measure of the security. Instead we would want to average over only the more likely distributions, since with overwhelming probability, the very rare ones will be few enough to be put in I_2. To avoid complicating matters further over a point which is only of minor importance we instead compute a ball park figure for Φ by using the expected values for β_{max} and β_{min}.

Table 1 shows some examples.

These figures clearly illustrate the disadvantage of using a random mapping. Look, for example, at the case $A = 100, B = 5, R = 0.5$. These parameters give us only 0.48 bits of security, and no other value of B does better. With a customized mapping we can get $0.5 \log 100 = 3.3$ bits so the use of a random mapping means a loss of 85 % of the security. Or, to put it another way, to get the same level of security we must make the user's password 6.6 times longer.[5]

Also note that some entries in the table above are negative. Adding such a password to an ensemble would make it *easier* for a computationally unbounded attacker to guess the codeword.

[5] If we look at $\Phi_{avg,(i)}$ rather than $\Phi_{worst,(i)}$, random mappings are handled better. The security is then $\Phi = (R-1)\log B - \log \beta_{max} + \log A$. Since β_{min} is not involved in this expression we do not have to fear empty buckets and can use a larger value for B. With $A = 100, R = 0.5$ we get the best security by using 45 buckets. This gives us 1.3 bits of security.