Knock Yourself Out: Secure Authentication with Short Re-Usable Passwords

Benjamin Güldenring, Volker Roth and Lars Ries
Computer Science
Freie Universität Berlin
Germany

Abstract—**We present Knock Yourself Out (KYO), a password generator that enables secure authentication against a computationally unbounded adversary. Master passwords can be surprisingly short and may be re-used for multiple service accounts even in the event of client compromises and multiple server compromises. At the same time, KYO is transparent to service operators and backwards-compatible. Master passwords are fully client-manageable while secrets shared with service operators can be kept constant. Likewise, secrets can be changed without having to change one's passwords. KYO does not rely on collision-resistant hash functions and can be implemented with fast non-cryptographic hash functions. We detail the design of KYO and we analyze its security mathematically in a random hash function model. In our empirical evaluation we find that KYO remains secure even if small sets of hash functions are used instead, in other words, KYO requires minimal storage and is highly practical.**

I. INTRODUCTION

Authenticating oneself to another party over a computer network is a fundamental security goal, and passwords are a pervasive mechanism to accomplish that goal. Passwords steadfastly resist being outmoded by other authentication technologies even though they are associated with bad security and bad usability. Common wisdom says that users must choose long passwords randomly for best security. Furthermore, users must not re-use passwords across different services or else one breach of a service may leak passwords that adversaries use to breach additional accounts. This is a realistic threat as the public press can testify. However, users cannot remember reliably many different secure passwords because of a variety of cognitive limitations and hence they tend to engage in insecure password practices. For example Florenĉio et al. [9] found that the average password is used at 4 sites. Bonneau and Preibusch [5] pointed out that after apparent improvements in password quality between the 70's and 90's, the web appears to have altered that trend again for worse.

NDSS '15, 8-11 February 2015, San Diego, CA, USA
Copyright 2015 Internet Society, ISBN 1-891562-38-X
http://dx.doi.org/10.14722/ndss.2015.23261

In our paper, we propose means to significantly enhance the security and usability of password security mechanisms against security breaches. Specifically, we address the following two risks associated with password mechanisms simultaneously.

1) An adversary obtains a copy of a user's personal password database and uses the information to log into that user's accounts. We call this a *client breach.*
2) An adversary obtains a copy of a server's password database and uses the information to log into accounts on other servers. We call this a *server breach.*

As this is done in related work (see section VII), we assume that the *integrity* of clients and servers has not been compromised, for example, by installing a key logger that intercepts entry of a master password. We do not address the risk that an adversary obtains a copy of a server's password database and uses the information to log into accounts on that same server. Instead, we discuss in Section VII how related work attempts to mitigate this risk under varying assumptions and how they blend with our proposals.

Existing work has addressed client breaches and server breaches largely in isolation, sometimes using similar mechanisms. For example, *decoy passwords* have been proposed as a means to trigger false login detection at servers in the case of client [4] and server [14] breaches. Although, no work we are aware of has addressed the case that passwords are shared among accounts and *multiple* password databases are breached. For example, intersecting two databases with decoy passwords quickly reveals all shared passwords and filters out decoys with high probability. This yields passwords that adversaries can potentially use to log onto first-party servers (breached) and third-party servers (not breached). Furthermore, the protection mechanisms proposed thus far have notable costs associated with them, for example, increased memory requirements [4], [14], [7], increased interactions and login latency [16], delayed verification [6] and varying amounts of required server changes [14], [16], [6]. Even *key stretching* [15] consumes energy that largely goes to waste, which is a best practice most password mechanism rely on in order to counter offline attacks. Lastly, none of these proposals introduce more user-friendly password choices.

In this paper, we propose *Knock Yourself Out* (KYO), a novel client-side password generator mechanism that mitigates the risks of simultaneous breaches of clients and multiple servers. The master password can be as short as four characters, and 10 characters if security against the breach of the password client and three servers is desired. Passwords can be re-used, that is, a client need only remember one master password. At the same time, KYO is fully client-manageable, that is, server-specific secrets can be changed without changing the master password, and the master password can be changed without affecting server secrets. KYO incurs no additional costs over naïve state of the art password mechanisms and it can be instantiated with fast non-cryptographic hash functions and without key stretching. We prove KYO secure in an information-theoretic setting, that is, in a random hash function model against unbounded adversaries. KYO is backwards-compatible with existing server implementations. In order to achieve its best security and usability product, KYO requires that servers are configured to lock accounts upon the first false login attempt. KYO verifies passwords locally and prevents false logins.

In what follows, we further detail our assumptions and threat model, followed by an overview over KYO's design, properties and features. Subsequently, we evaluate KYO mathematically in a random hash function model and we briefly point out the impact of moving from a random hash function to smaller sets of functions. In our subsequent empirical exploration, we find that even small sets of functions exhibit the necessary and sufficient properties we require. Based on our theoretical analysis, we estimate the salient properties of KYO, for example, the relationships between password length and the number of client and server breaches against which passwords shall be secure. We discuss obvious and less obvious application areas for KYO, compare KYO to related work and end with conclusions and an outlook of what kind of future work we deem most exciting.

II. Assumptions and Threat Model

For ease of description we assume throughout the paper that Alice communicates with Bob, Carol and David. One may think of Alice as a user with a client computer, and of Bob, Carol and Dave as servers. Alice shares a secret with Bob, and another one with Carol, and another one with David. KYO allows Alice to generate all of her shared secrets based on a single password, if she so desires. For the sake of generality, we consider the scenario that Alice wishes to use one password for Bob and a different one for Carol and David. More formally, let ρ_1, ρ_2 be Alice's passwords, let $\gamma_1 \ldots, \gamma_3$ be the secrets that Alice shares with the other parties and let $\sigma_{A \to B}$ be a *seed* Alice uses to generate the secret she shares with Bob. Let F be the generator function, then:

$$\gamma_1 = F_{\sigma_{A \to B}}(\rho_1) \qquad \gamma_2 = F_{\sigma_{A \to C}}(\rho_2)$$
$$\gamma_3 = F_{\sigma_{A \to D}}(\rho_2)$$

This will be our running example. Alice authenticates herself to Bob by presenting γ_1 to Bob in a fashion that assures that the presentation is fresh and that γ_1 is not revealed to outside parties. In practice, Alice may connect to Bob using a TLS connection, Bob authenticates itself to Alice based on a valid X.509v3 certificate, and Alice transmits her shared secret to Bob in order to authenticate herself to Bob. All parties store their seeds and corresponding identities.

A. Power of the adversary

The adversary may compromise any number of the parties, for example, Alice, Bob and Carol. We assume that none of the compromised parties store their passwords, and that they do not enter their passwords following a compromise. This means that, for example, the adversary learns all secrets Bob shares with Alice, and all seeds Alice stores, but not the password of Alice. Furthermore, we assume that the adversary is computationally unbounded, that is, he can compute as much as he likes.

The risks we outlined encompass a wide variety of actual risks to which Alice may expose herself. For example: Alice looses her laptop, someone seizes or steals her laptop, or someone compromises one or multiple services that Alice uses and steals any available secret information on the users of the service.

B. Security and safety objectives

We say that an adversary *succeeds* if he authenticates himself as Alice to an uncompromised party. Depending on which parties the adversary compromises and how many of them, we seek to make statements of the form *the success probability of the adversary is* ϵ, where ϵ is a concrete probability based on some choice of security parameters. We will call the probability ϵ *insecurity* and accordingly $1 - \epsilon$ *security*.

In an analoguous fashion, we wish to make statements about the *safety* of password entry. Jumping slightly ahead, KYO verifies correct password entry locally. Therefore, one might assume that Alice, the user, has an infinite number of tries and, consequently, Bob will never lock out Alice because Alice only submits correct secrets to him. However, if Alice has an infinite number of tries then an adversary who breaches the client can brute-force the mechanism offline. Since we wish to provide security against client breaches, we cannot allow this to happen. For this reason, KYO takes a different approach and as a consequence there exists a probability δ with which KYO accepts a false password as correct. In this case, KYO will produce a wrong secret and Bob will lock Alice out. Hence, security determines an upper bound on safety. In our mathematical analysis, we show how to control ϵ and δ. In our writing, we refer to δ as *insafety* and to $1 - \delta$ as *safety.*

C. Baseline security

Eventually, we must choose concrete parameters and these parameters determine the actual security and safety of KYO and the necessary lengths of passwords. We want our passwords to be as short as possible without sacrificing security and safety. Hence the question arises what are acceptable levels of security and safety. Towards guidance we look at the security of ATM machines. It is probably safe to assume that the risk of unauthorized access to one's bank account is an acceptable baseline. We further assume that the adversary is in possession of a banking card and hence the remaining security lies in the PIN. ATMs typically limit the number of invalid PIN entries to two so that a user has three attempts to enter the correct PIN. Conservatively, we assume that the PIN number is chosen uniformly at random from the set $\{0, \ldots, 9\}^4$ and hence there are $N = 10^4$ equally likely PINs. The failure probability of guessing a particular PIN in n tries is then given by the hypergeometric distribution

$$\binom{1}{0}\binom{N-1}{n}\cdot\binom{N}{n}^{-1},$$

which simplifies to $1-n\cdot 10^{-4}$. Thus for three allowed PIN entries the insecurity of this scheme is $3\cdot 10^{-4}$. Since ATM machines are in widespread use, we believe $3\cdot 10^{-4}$ to be a reasonable baseline for insecurity. Likewise, it is probably fair to argue that, most of the time, a security breach in this scenario is a more serious event than a lockout. From this we conclude that the security bound is suitable as a bound for safety as well.

D. Residual risks and attacks

We have mentioned before that KYO performs best if servers lock an account immediately if the client provides a false secret. This poses a denial of service risk. Actually, this risk exists in any password system that tolerates only a low constant number of false logins. A second risk is that an adversary succeeds simply by fixing a random password and by trying this password on a large number of user accounts. If a server has a large number of users then with a good probability, the adversary will breach some accounts. Again, this is a risk that applies to password systems in general. Although, the risk is somewhat higher in the case of KYO because KYO uses short secrets in addition to short passwords. Short secrets have other security benefits, as we discuss in following sections. Fortunately, there is a general solution that mitigates the two risks we just mentioned. Randomizing users' identities forces the adversary to guess matching user identities and passwords. Hence, we can augment n bits of password entropy with an arbitrary entropy m from user identities for a total of $n+m$ bits. Florencîo et al. [10] discuss this approach nicely in a paper of theirs and therefore we assume henceforth that the problem is solved adequately. We discuss another similar solution in Section VI to make KYO backwards compatible with existing systems.

III. Design

A contemporary password check fulfils two goals. First, it verifies that Alice entered her password *correctly,* and second, it verifies that the entered password is *authentic.* The former is a *safety check* (that the password was entered correctly) and the latter is a *security check* (it matches the one associated with the user account). If the server verifies both safety and security, this distinction seems artificial since the response is typically the same in both cases (for example, a prompt to re-enter one's password). It does not make sense to distinguish between them. In the case of our password generator schemes there is a distinction, however, because Alice checks her password ρ_1 *locally* for correctness, and Bob stops accepting communication from Alice *immediately* if Alice presents anything but $\gamma_{A\to B}$ as her secret. In order to prevent this from happening when she enters a wrong password, Alice uses an error detecting code in order to check passwords for correctness locally. This code will however detect only a fraction of all possible false password entries.

Towards an understanding of the principles that underpin our KYO password generator, consider a function F_σ that

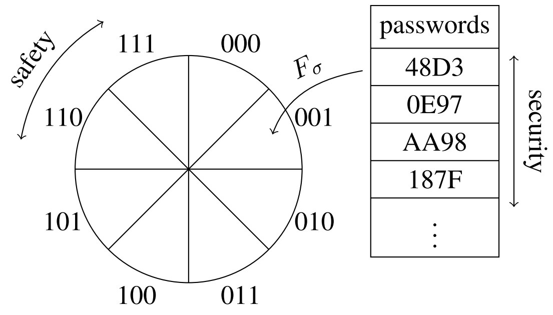

Fig. 1. Example password partition among eight subsets. As the number of subsets increases, the list of colliding passwords becomes smaller.

partitions the set of all hexadecimal passwords of length four into eight evenly sized subsets as indicated in Figure 1. For example, F_σ maps the password $(48\mathrm{D}3)_{16}$ to the subset $(001)_2$. We use the output of F as the error correction information and store it locally on a computer. Assume that F_σ maps passwords randomly to each subset. It follows that the probability that any password is in subset $(001)_2$ is $1/8$. The *safety* of F is therefore the probability that a password other than $(48\mathrm{D}3)_{16}$ is *not* in that same subset, which is $1-1/8=0.875$. Assuming that passwords are chosen uniformly at random, each subset contains $(16^4/2^3)=8192$ equally likely passwords. Therefore, the security of F against an attacker with knowledge of F and the subset $(001)_2$ is $1-1/8192\approx 0.9999$, even if the adversary is unbounded.

There is a small catch, though. If Alice enters her password incorrectly multiple times and her entries are pair-wise different then safety decreases because the remaining number of incorrect passwords not mapped to $(001)_2$ decreases compared to the number of incorrect passwords mapped to $(001)_2$. This effect compounds over multiple authentication sessions unless Alice always makes the same errors. Therefore, KYO chooses a new seed σ each time Alice enters her password correctly. This renders authentication sessions independent of each other and improves safety. In what follows, we explore these ideas further. Particularly, we will use two instances of the sketched function F, one for checking safety and another one to generate secrets.

A. KYO families

We require functions with certain properties for KYO. In this section, we define these requirements. We begin by introducing necessary terminology. Towards this end, let $S=\{0,1\}^s, P=\{0,1\}^n, L=\{0,1\}^\ell$ with $s\geq n>\ell$ be sets and let $(F_\sigma)_{\sigma\in S}$ be a family of hash functions with

$$F_\sigma : P\to L, \qquad \gamma = F_\sigma(\rho).$$

We call $\rho\in P$ *password* and $\sigma\in S$ *seed.* We call $\gamma\in L$ *(shared) secret* if it is used for authentication purposes, or *digest* if it is used for error detection. If there is no need for a distinction between these two cases then we use the term *digest* for both. Note that since we require that $n>\ell$, F_σ will be a lossy function, that is, non-injective. Unless noted otherwise, whenever we say that some element is chosen randomly from some set, we mean that it is chosen uniformly at random from that set and independent from any other values that might be chosen from the same set at the same time.

pw	pw-info
1	(σ_1, γ_1)
2	(σ_2, γ_2)

Password table PW

entry	pw	seed
Bob	1	$\sigma_{A\to B}$
Carol	2	$\sigma_{A\to C}$
Dave	2	$\sigma_{A\to D}$

Address book T

Fig. 2. Contents of Alice's address book and password table for a shared password ρ_2 for Carol and Dave, and a separate password ρ_1 for Bob.

Requirement 1 (Password selection)**.** *Given arbitrary passwords $\rho_1, \ldots, \rho_N \in P$, seeds $\sigma_1, \ldots, \sigma_N \in S$ and digests $\gamma_1, \ldots, \gamma_N \in L$ with $F_{\sigma_i}(\rho_i) = \gamma_i$, there exists a password $\rho \in P$ with $F_{\sigma_i}(\rho) \neq \gamma_i$ for all $i = 1, \ldots, N$.*

Requirement 2 (Seed selection 1)**.** *Given arbitrary passwords $\rho, \rho_1, \ldots, \rho_N \in P$, there exists a seed $\sigma \in S$ with $F_\sigma(\rho) \neq F_\sigma(\rho_i)$ for all $i = 1, \ldots, N$.*

Requirement 3 (Seed selection 2)**.** *Given an arbitrary password $\rho \in P$ and a shared secret $\gamma \in L$, there exists a seed $\sigma \in S$ with $F_\sigma(\rho) = \gamma$.*

We also require that there exist algorithms that compute passwords and seeds according to these requirements. We do not require that these algorithms are efficient in a complexity-theoretic sense but instead that running them is feasible for practical choices of parameters. In Section IV-A, we show that random sampling computes passwords and seeds with high probability.

B. Data structures

The KYO password generator manages and generates secrets based on a *password table* PW and an *address book table* T, see Figure 2 for illustration. The password table keeps error detection information on passwords, that is, a seed and digest for each password. The address book contains records that name the recipient (for example, a server), the index of the associated password record in the password table, and a seed specific to that recipient and the associated password. In what follows, we describe how KYO implements common operations such as verifying passwords, computing secrets, adding and removing accounts, adding and changing passwords, et cetera.

C. Password creation and verification

In order to create a new password entry in PW, Alice first selects a random password $\rho \in P$. If the password table PW is still empty, she then selects a random seed $\sigma \in S$, calculates $\gamma := F_\sigma(\rho)$ and stores $(\sigma_1, \gamma_1) := (\sigma, \gamma)$ in PW. If PW is not empty then Alice selects a random $\rho \in P$ that meets requirement 1 and a random seed $\sigma \in S$ that meets requirement 2. Next, she calculates $\gamma := F_\sigma(\rho)$ and stores $(\sigma_i, \gamma_i) := (\sigma, \gamma)$ with a fresh index i in PW. This ensures that table PW satisfies the following constraint with a high probability:

Constraint 1 (Confused passwords)**.** *Let ρ_i be Alice's passwords. Table PW satisfies the confused passwords constraint iff for all i and j:*

$$F_{\sigma_i}(\rho_j) = \gamma_i \Rightarrow j = i.$$

Scheme 1 KYO secret derivation

```
procedure AUTH(Password $\rho$, Recipient $R$)
    $r$ := T[R].pw
    $(\sigma_{\text{check}}, \gamma_{\text{check}})$ := PW[$r$].pw-info
    $\sigma$ := T[$R$].seed

    if $F_{\sigma_{\text{check}}}(\rho) = \gamma_{\text{check}}$ then
        $\sigma' \leftarrow^R S$                          ▷ fulfilling requirement 2
        PW[$r$].pw-info := $(\sigma', \gamma')$

        return $F_\sigma(\rho)$                      ▷ shared secret derivation
    else
        return $\perp$                          ▷ report error and restart
```

In order to verify if a password ρ is the correct password for entry i, Alice retrieves the entry (σ_i, γ_i) from PW and tests whether $F_{\sigma_i}(\rho) = \gamma_i$. If so, we say that ρ *verifies* under (σ_i, γ_i). Constraint 1 ensures that each password verifies under only one entry (σ_i, γ_i) in PW, thus providing *password safety*.

D. Adding recipients and shared secret derivation

In order to add a recipient, say Bob, to the address book T, Alice selects a random seed $\sigma \in S$ and chooses a password index i from PW. Alice then enters her password ρ. If ρ verifies under (σ_i, γ_i) then Alice stores the entry $\langle \textbf{Bob}, i, \sigma \rangle$ in T. She finally transmits the shared secret $\gamma := F_\sigma(\rho)$ to Bob who will store it locally in order to verify Alice with it in the future. Figure 2 shows Alice's tables PW and T for our running example of one password for Bob and a shared one for Carol and Dave.

In order to derive a shared secret for recipient R, Alice enters a password ρ and runs Algorithm AUTH(ρ, R) in Scheme 1. Upon entering ρ, the algorithm tests if ρ verifies for recipient R. If the test fails then the algorithm fails and Alice is assumed to have entered a wrong password. If the test succeeds then the algorithms updates the corresponding entry in PW and outputs a shared secret. As we show in Section IV, the update of the entry in PW is important for the scheme's safety.

E. Changing passwords and shared secrets

Alice might want to change a secret she shares with Bob without changing her password. This is not possible with contemporary password generators but it is straightfoward in KYO. Alice selects a new random seed $\sigma \in S$, calculates the shared secret and informs the recipient of it's new value.

Changing one's password while keeping all shared secrets the same is more involved. Let σ_i, γ_i with $i \in I$ be the seeds and shared secrets that are protected by password ρ (the seed is taken from table T), that is, it holds that

$$\forall i \in I : F_{\sigma_i}(\rho) = \gamma_i.$$

Alice chooses a new random password $\rho' \in P$ that satisfies Constraint 1. Subsequently, she must find seeds $\sigma'_i \in S$ that fulfill the equation before. Requirement 3 states that it is possible to find such seed values. Once found, Alice replaces the entry in PW with $(\sigma', F_{\sigma'}(\rho'))$ with a fresh selected seed $\sigma' \in S$ and updates the entries of T with the found seeds σ'_i.

In our evaluation in Section IV we show that it is not only possible to find these seeds σ'_i but also that doing so does not compromise security. We will call this scenario *selected seeds,* because a seed is "selected" to fit an independently chosen password and shared secret.

F. Managing password sharing

Alice may use the same technique to transparently manage groups, that is, the recipients for whom she uses the same password, say ρ. In order to include someone in that group with whom she shares secret γ, all she needs to do is find an additional σ so that $F_\sigma(\rho) = \gamma$.

In order to exclude someone from a group, she simply chooses a new random password ρ' that satisfies constraint 1 and finds a seed σ that satisfies $F_\sigma(\rho') = \gamma$. As before, Requirements 1 and 3 ensure that this is possible.

G. Synchronizing shared secrets

In what we have seen so far, Bob stores the secret γ he shares with Alice locally in order to be able to verify if a client is indeed Alice. Instead of storing the secret explicitly, Bob can protect γ using F in the same way Alice protects her secrets. In order to do so he selects a random verification password $\rho_v \in P$ and finds a verification seed $\sigma_v \in S$ so that $F_{\sigma_v}(\rho_v) = \gamma$. Once found, he stores σ_v in a separate column in his own address book T. The necessary Algorithm AUTH_v for this is basically the same as AUTH in Scheme 1, with the symbols of seeds and digests substituted in the obvious manner. Bob runs $\text{AUTH}_v(\rho_v, R)$ and accepts if the output is the same as the shared secret he received from Alice.

If Bob had a password beforehand that he uses to authenticate himself to Alice then Bob can keep his password and synchronize the secret he shared with Alice with the secret that Alice shared with him. Essentially, both Alice and Bob choose passwords and seeds that map to the same secret. In this fashion, Bob des not have to store two seeds for two secrets both shared with Alice. Since passwords and shared secrets are chosen randomly and independently of each other, this scenario is essentially the same as the *selected seeds* scenario and Requirement 3 ensures that it is possible to find the necessary seed values.

It is worth noting that if all participants used synchronized secrets then an adversary would not be able to impersonate anyone even if he had access to *all* KYO tables of everyone, as we show in our evaluation. The downside is that synchronized secrets link the security of the authentication of Bob and Alice. This means that if an adversary ever learns the secret that Alice uses to authenticate herself to Bob (for example, by breaking into their communication channel) then the adversary cannot only authenticate himself as Alice to Bob but also vice versa.

H. Summary of properties

By what we have described so far, it should be clear that Alice is free to manage groups of users for whom she can choose same or different passwords as a means of protecting shared secrets. Furthermore, Alice is free to change passwords and secrets as she pleases. Moreover, all communicating parties can synchronize their seeds so that they can choose individual passwords to manage mutually shared secrets for authentication purposes. The use of shared secrets is not limited to online communication, though. Since KYO does not require a feedback channel, communicating parties may use their shared secrets for other purposes as well, for example, for the purpose of authenticating electronic mail. It is important to note, though, that the shared secret is short by design and, hence, the authentication scheme must be designed carefully to account for this property.

IV. Evaluation

In this section we analyze KYO mathematically based on the assumption that F is a random hash function. This implies that $|S| = |L|^{|P|}$. We update our definition of KYO families from Section III-A accordingly as follows:

Definition 1 (KYO family)**.** *Let F be the set of all functions mapping P to L and let S be an index set enumerating them. Then $(F_\sigma)_{\sigma \in S}$ is called a KYO(n, s, ℓ)-Family.*

The definition implies that F_σ is selected randomly from F given a random seed σ. For growing P and L, the length of a seed quickly exceeds what is feasibly processed and stored on a computer. In practice, we will therefore work with index sets that are much smaller. This is unproblematic as long as the number of recipients per password is small compared to 2^s. This is because the distribution over random samples of the indexes has the same mathematical expectation as F. Therefore, we just need "enough" indexes so that F meets the requirements we introduced in Section III-A. The analysis and the proofs are simpler in the case of a random hash function, though. We have proven corresponding lemmas and theorems for random index subsets as well but presenting them here while meeting the page limit would render the presentation unreadable.

In what follows, we show that KYO families meet the requirements we put forward in Section III-A. For ease of reading we only state the results in this section and give the proofs in Appendix A. Subsequently, we analyze the *safety* of KYO families followed by an analysis of the *security* of KYO families.

A. KYO generator properties

For a given $\rho \in \{0,1\}^n$ and $\sigma, \in \{0,1\}^s$, we denote the set of passwords that map to γ under σ with

$$F_\sigma^{-1}(\gamma) := \{\rho' \in P \mid F_\sigma(\rho') = \gamma\}.$$

We now show that a KYO family meets Requirements 1 and 2 and 3 with high probability. The following lemmas state that random sampling is sufficient to achieve *safety* with high probability.

Lemma 1 (Password creation)**.** *If the password table PW already contains N passwords, the number of random tries to find the $(N+1)$-th password is geometrically distributed with average value $(1 - N \cdot 2^{-\ell})^{-1}$.*

Lemma 2 (Seed selection 1)**.** *For $N-1$ given passwords, a randomly chosen seed σ meets Requirement 2 with probability $(1 - 2^{-\ell})^{N-1}$.*

The design of the KYO generator assumes that it is possible to find seeds $\sigma_1, \ldots, \sigma_N \in S$ for any given combination $\rho, \gamma_1, \ldots, \gamma_N$ efficiently. The following lemma states that this is possible and how long it takes to find a (random) σ with $F_\sigma(\rho) = \rho$.

Lemma 3 (Seed selection 2). *Given a password ρ and shared secrets $\gamma_1, \ldots, \gamma_N$, the number of random tries to find seeds $\sigma_1, \ldots, \sigma_N$ that satisfy $F_{\sigma_i}(\rho) = \gamma_i$ is geometric distributed with average value $N \cdot 2^\ell$.*

Even though the last result is exponential in ℓ the value of ℓ is usually small enough for "interesting" levels of security and safety so that this is indeed feasible, see Section IV-D.

B. Safety evaluation

The safety constraints must ensure that a password entered incorrectly does not verify under a given seed. More formally, assume that Alice's password is ρ and that she enters incorrect passwords $\rho_1, \ldots, \rho_N \neq \rho$. The goal of the analysis is to determine the probability that one of the ρ_i yields the same digest as ρ. We can approach this problem from two directions in principle. We can compute probabilities over the distribution of passwords or over the distribution of seeds. However, we do not know how password input errors are distributed since this is highly dependent on the user and perhaps the password. Therefore, we compute probabilities over the seeds. A technical detail is that we must assume that passwords and seeds are independent of each other, which requires justification. Indeed, we expect that users do not "see" seeds and hence do not make input errors that depend on knowing a particular seed. We believe this is reasonable and we proceed with a formal definition of our safety experiment.

Definition 2 (Safety experiment). *The safety experiment is given by the probability space $(\Omega, \mathcal{P}(\Omega), \Pr)$, with the set of elementary events*

$$\Omega = \{\sigma \in S\},$$

and $\mathcal{P}(\Omega)$ being the power set of Ω and $\Pr$ being a function that maps each event in $\mathcal{P}(\Omega)$ to its probability.

We give a formal definition of the term *safety* next. Given a digest γ and incorrectly entered passwords $\rho_1, \ldots, \rho_N$ we look for the probability that at least one ρ_i yields the same digest as the correct password ρ. The following definition captures this, while taking the probability over the choices of σ as discussed before.

Definition 3 (Safety). *Given $\gamma \in L$ with $|L| = 2^\ell$, $\rho \in P$ and $\rho \neq \rho_1, \ldots, \rho_N \in P$,* safety *is defined as the probability*

$$\Pr[\{\sigma \in \Omega : \wedge_{i=1}^{N} F_\sigma(\rho_i) \neq \gamma\}].$$

Finally, we give an approximation of this probability for the case that N is small compared to 2^s.

Lemma 4 (Safety). *Let N be the number of pairwise distinct wrongly entered passwords and let N be small compared to 2^s. Then the safety of the KYO scheme is binomial distributed with average value*

$$1 - (1 - 2^{-\ell})^N .$$

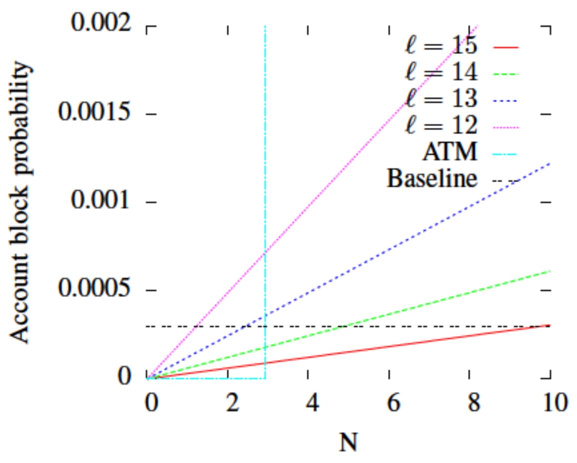

Fig. 3. Plots the probability that an account is blocked given that N wrong passwords are entered according to Lemma 4. The vertical line represents the three strikes rule of ATM PIN entry. The horizontal baseline represents the security of ATM PIN entry, see also Section II-C. For $\ell = 15$ and $N = 8$ it is more likely that an adversary guesses a correct ATM PIN than that a user is locked out while using KYO.

Note that the probability of safety decreases as input errors accumulate over repeated authentications (unless users always make the same input errors, which we cannot assume). This is the reason why KYO chooses a fresh digest γ and seed σ once the user enters her password correctly (the correct password is needed for this). In this way it is guaranteed that the experiment "restarts" regularly.

Figure 3 illustrates KYO's safety for different values of ℓ and contrasts them with the safety and security of ATM PIN entry. The vertical line represents the safety of the three strikes rule of ATM PIN entry. The ATM account is guaranteed to be blocked after three incorrect PIN entries and is never blocked with fewer incorrect entries. The horizontal baseline represents the security of ATM PIN entry, that is, the success probability of an adversary in a guessing attack with the three strikes rule. The plots show, for example, that the safety of KYO for $N = 8$ and $\ell = 15$ is better than the security of ATM PIN entry for $N = 3$.

C. Security evaluation

We consider an attacker that tries to impersonate Alice against a fixed but arbitrary recipient. We call the password Alice uses for this recipient the *target password* and the corresponding seed and secret the *target seed* and *target secret.* In order to impersonate Alice, the attacker has to predict the target secret exactly because secrets are compared for equality. We measure his success probability indirectly using a probability distribution $\Pr[\Gamma = \gamma \mid K]$, where K denotes any knowledge the attacker has gained and Γ is a random variable for the target secret. Thus $\Pr[\Gamma = \gamma \mid K]$ is the probability that the target secret is γ given the conditions K. Note that utilizing only this distribution, an attacker may never succeed with any probability greater than

$$\max\{\Pr[\Gamma = \gamma_i \mid K] : \gamma_i \in \Gamma\}.$$

For example: if $K = \emptyset$ then any shared secret γ chosen by an attacker will be correct with a probability of $|\,\Gamma\,|^{-1}$. Recall that KYO does not allow unsuccessful authentication attempts, that is, an attacker has only one try to predict the target secret. If Alice re-uses the target password for $N-1$ other recipients then an adversary may try to authenticate to each of them to learn information about the target password. In order to simplify this part of our analysis we assume that an adversary is then given the secrets of those $N-1$ recipients. From here on, we assume that K contains the following types of information:

- The contents of Alice's address book,
- a set of passwords,
- a set of shared secrets other than the target, and
- whether any shared secrets are selected.

In practice, K might include information from sources outside of our model as well, for example, partial information on the target password derived from smudges on the touch screen of a smartphone [2]. We consider these side-channels out of our scope. For the purpose of our analysis, the items above are a complete characterization of the types of information an adversary can have. Hence, by taking into account all combinations of these items, we get an exhaustive view on an adversary's success probability.

We begin by reasoning that the adversary must know a user's address book to glean information on a target secret. Assume he does not. Then he may know at most all secrets other than the target secret and he may know all passwords. The secrets yield information on their corresponding passwords but passwords only yield information on other passwords through digests in the password table. Therefore, the best the adversary can hope is to learn the target password, which we just assumed he already knows. Since F comprises all functions from P to L, there exist equally many seeds σ for any pair of ρ and γ so that $F_\sigma(\rho) = \gamma$. This follows from a simple labeling argument. Hence, given the target password but not the target seed, all secrets are equally likely, which completes our argument.

In what follows, we assume that the adversary has obtained information from the address book table and the password table. We are left with two types of information transfers to analyze:

1) The first transfer is from secrets to passwords through the contents of the password table.
2) The second transfer is from passwords to secrets through the contents of the address book table.

Remember that the goal of the adversary is to glean information on the target secret based on at most a fixed number of other leaked secrets and information in the user's password table and address book table.

We begin with the second type of transfer. Our analytical goal is to measure the probability distribution over secrets given partial knowledge about a password. Note that the information whether the target seed is selected (as opposed to having been chosen entirely at random) does not help here since the number of seeds to choose from are the same for every password and digest combination. Therefore, we ignore selected seeds for now. The following definition models the secret guessing experiment accordingly.

Definition 4 (Guessing secrets experiment)**.**
The secrets guessing experiment is defined by the probability space $(\Omega, \mathcal{P}(\Omega), \Pr)$*, with*

$$\Omega := \{(\rho, \sigma) \in P \times S\}.$$

In order to work with this definition we define the random variables R that gives the target password, Σ that gives the target seed and Γ the target secret. In detail, for given ρ, σ or γ:

- $R = \rho$ denotes all events (ρ, σ) with $\sigma \in S$,
- $\Sigma = \sigma$ denotes all events (ρ, σ) with $\rho \in P$, and
- $\Gamma = \gamma$ denotes all events $(\rho, \sigma) \in \Omega$ with $F_\sigma(\rho) = \gamma$.

As we will see in the next section, if an adversary learns secrets and seeds he will learn a candidate set C of passwords that are all equally likely. The next lemma gives the guessing probability with knowledge of C.

Lemma 5 (Guessing secrets)**.** *Let C be a set of equally likely candidate passwords and let $c = |C|$. We define $M = F_\sigma(C)$. Then the probability*

$$\Pr[\Gamma = \gamma \mid \Sigma = \sigma \wedge \rho \in C]$$

is binomial distributed with the average value

$$\left(2^\ell \cdot (1 - (1 - 2^{-\ell})^c)\right)^{-1}$$

for every $\gamma \in M$.

We discuss the first type of information transfer next, that is, how an adversary may gain knowledge of a set of candidate passwords C. Specifically, we investigate what happens if the adversary learns the contents of Alice's address book and her shared secrets. We obtain results about the size of the password candidate set C and that all passwords within C are equally likely. These results and the results from Lemma 5 together yield the overall guessing probability.

Recall that in the selected seeds scenario, Alice fixes a password ρ and a secret γ_i and then chooses a seed σ_i so that $F_{\sigma_i}(\rho) = \gamma_i$. Let $S_i(\rho)$ denote the set from which Alice chooses the seed σ_i. If Alice does not fix a secret γ_i then let $S_i(\rho)$ denote the set of all seeds, that is, $S_i(\rho) = S$. Next, we model the probability distribution over passwords if seeds and secrets are learned.

Definition 5 (Guessing passwords experiment)**.**
The password guessing experiment is defined by the probability space $(\Omega, \mathcal{P}(\Omega), \Pr)$*, with*

$$\Omega = \{(\rho, \sigma_1, \ldots, \sigma_N) \mid \rho \in P, \sigma_i \in S_i(\rho)\},$$

and

$$\Pr[(\rho, \sigma_1, \ldots, \sigma_N)] := \frac{1}{\mid P \mid} \cdot \prod_{i=1}^{N} \frac{1}{\mid S_i(\rho) \mid},$$

where

$$S_i(\rho) := \begin{cases} \{\sigma \mid F_\sigma(\rho) = \gamma_i\} & \textit{a digest } \gamma_i \textit{ is given} \\ S & \textit{else.} \end{cases}$$

In addition to the random variables we have defined before $(R, \Sigma$ and $\Gamma)$ we need random variables Σ_i and Γ_i. Variable Σ_i describes the seed for the i-th pair and variable Γ_i describes

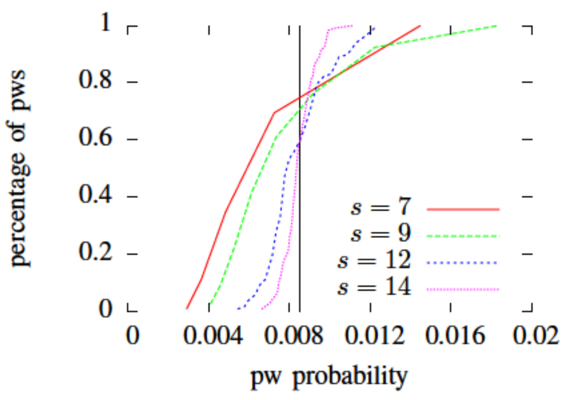

Fig. 4. Shows the effect of seed length on the cumulative probability distribution of passwords if a seed is selected. The vertical line denotes the ideal distribution, that is, for $s = \ell \cdot 2^n$.

the corresponding secret. More precisely, we define for given σ and γ:

$$\{\Sigma_i = \sigma\} := \{(\rho, \sigma_1, \ldots, \sigma_n) \in \Omega \mid \sigma_i = \sigma\}$$
$$\{\Gamma_i = \gamma\} := \{(\rho, \sigma_1, \ldots, \sigma_n) \in \Omega \mid F_{\sigma_i}(\rho) = \gamma\}$$

Note that by learning the contents of Alice's address book, the adversary already knows the seed and digest that are used for error detection. Hence, $N \geq 1$.

The question is what an adversary learns about ρ if he learns seeds σ_i and corresponding secrets γ_i with $F_{\sigma_i}(\rho) = \gamma_i$. Let

$$M := \cap_{i=1}^{N} F_{\sigma_i}^{-1}(\gamma_i)$$

be the set of all passwords that do not contradict the knowledge the adversary has gained so far. An adversary obviously knows that ρ is in M but within M all passwords are equally likely. For the size of M we get the following result.

Lemma 6. *The size of M is binomial distributed with average value $2^{n-N\cdot\ell}$.*

We now arrive at the main result in this section, which states the guessing distribution of passwords if seeds and digests are known.

Theorem 1 (Guessing passwords). *It is*

$$\Pr[R = \rho \mid \wedge_{i=1}^{N}(\Sigma_i = \sigma_i \wedge \Gamma_i = \gamma_i)] = \begin{cases} M^{-1} & \rho \in M \\ 0 & \textit{else.} \end{cases}$$

with M binomial distributed with average value $2^{n-N\cdot\ell}$.

D. Discussion and case studies

In practical implementations where S does not cover all functions mapping from P to L, a short seed length has a significant influence on the guessing probability because the passwords within M are not equally likely. Figure 4 illustrates the effect for an implementation based on the SHA-256 function. The vertical line at $\simeq 0.008$ marks the probability distribution that results from choosing the seed uniformly at random from the set of all functions, that is, all passwords are equally likely. The other graphs show the cumulative distribution of password probabilities for varying seed lengths, a password length of 14 bits, a digest length of 7 bits and arbitrarily chosen digest 12 and target seed 123. Evidently, the distributions quickly approach the ideal. In practice, one would use somewhat larger seeds but producing the plots in Figure 4 takes computation in the order of the size of the seed space, which quickly becomes computationally infeasible. Therefore, we evaluated the distributions only up to $s = 14$. A practical heuristic that worked well in all of our tests is to set the seed length to the sum of the password length and the digest length. The important point is to not make the seeds shorter than the passwords.

The length of shared secrets should be at least 12 bits in order to achieve security that is better than the baseline we motivated in Section II-C. The reason is simply that 12 is the smallest integer so that

$$1 - 2^{-12} \geq 1 - 3 \cdot 10^{-4} = 0.9997$$

where the right-hand side is the security of guessing an ATM PIN in three attemps. Hence, we assume that $\ell = 12$ for secrets. For digests, we assume $\ell = 13$ so that our safety is at least as good as our security baseline for up to 3 password entries, as the graphs in Figure 4 show.

Based on Lemma 6 we can now estimate how long a password must be so that it resists the disclosure of N *pairs*. A *pair* is a seed and a digest, or a seed and a shared secret. Seeds and digests must obviously result from client breaches whereas shared secrets must result from server breaches. Resistance against the disclosure of N pairs therefore means a client breach and $N - 1$ server breaches. Without a client breach, and hence without seeds, the adversary has no pairs to his advantage. However, while Lemma 6 informs us only about the security of a password against password guessing we must also take into account that two passwords may be mapped to the same shared secret by two different seeds. In other words, password guessing security is not a lower bound of secret guessing security. Hence, the password length must be chosen so that the secret guessing probability given in Lemma 5 is below the baseline as well. In practice, the difference between the two lengths is about a bit. Table I reports the password length necessary to beat the baseline in terms of guessing passwords and guessing secrets for a varying number of disclosed pairs. In addition to bit length, we consider three categories of passwords:

1) Lower-case letters and numbers
2) Lower-case and upper-case letters and numbers
3) Printable ASCII characters without space and `' \ " ^ ~`

For example, if Alice uses a password for four servers and if she expects her client to be breached then she should choose a password length that renders her password secure against four disclosures of a pair (1 client and 3 servers). The reason is simply that it does not make sense to protect the password against the fifth disclosure because all her accounts have been breached by that time anyhow. In order to protect herself with better than baseline security she needs either a 4 character ASCII password or a password of length 13 that consists only of lower-case letters and digits.

This compares favorably with practice. Florenĉio et. al. [9] found that an average client has about 6.5 distinct passwords, and each passwords is used on average for about 3.9 sites.

pairs (i)	lower-case	alphanumeric	ASCII	bits
1	6	5	4	25.06
2	9	7	6	38.06
3	11	9	8	51.06
4	14	11	10	64.06
alphabeth	36	62	90	2

TABLE I. REPORTS THE MINIMUM PASSWORD LENGTH FOR BETTER THAN BASELINE SECURITY ASSUMING THE DISCLOSURE OF i PAIRS (σ, γ).

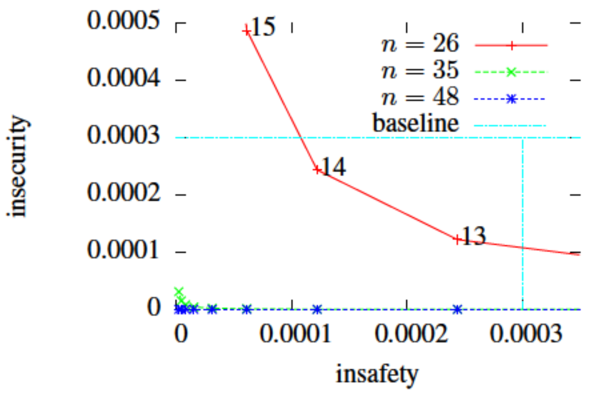

Fig. 5. Illustrates the insafety versus insecurity trade-off for password length n bit and example digest lengths. A choice of $n = 26$ and digest length 13 beats the ATM security baseline in both dimensions.

Their data includes some over-counting, which means that the actual amount of password sharing may be smaller. They found that average passwords have a bitstrength of about 40.54 bit or $6 - 7$ alphanumeric characters. However, they ignored passwords with a bitstrength less than 30. They also reported that the majority of users choose lowercase passwords only, unless forced otherwise. It seems that KYO is well aligned with current password practice. For passwords that are slightly longer than average ones, her unbreached secrets are secure against unbounded adversaries even if her client is breached and three servers that she uses.

There exists a trade-off between safety and security. Longer digests improve safety but they yield more information on the password. Hence, a longer password is required in order to achieve the same security and greater safety. Figure 5 illustrates the trade-off. The axes measure the insecurity and insafety, and the graphs represent choices of password lengths, and the points on the graph represent digest lengths from 12 to 20 bits. For readability, we have labeled only the points on the graph for password length 26 bits. As can be seen, a choice of 13 bits for the digest length keeps the insecurity and insafety below the baseline.

V. IMPLEMENTATION

We compared two implementations of a KYO Family. One implementation uses the SHA256 cryptographic hash function and the other implementation uses simple 2-independent universal hash function.

A. SHA-256 hash function

Our implementation builds upon the SHA-256 implementation by Oliver Gay.[1] The hash is calculated as

$$F_\sigma(\rho) = \text{truncate}_{32}(\text{SHA}_{256}(\sigma || \rho)) \bmod 2^\ell$$

with $||$ denoting the binary concatenation operator, σ padded to s bits and ρ padded to n bits. The truncate_{32} operator truncates the result to the first 32 bits from the left, which are interpreted as an unsigned big endian number.

B. 2-Universal hash function

The 2-universal (2-Univ) hash function family is based on the design of Wegman and Carter [25] and is calculated as

$$F_\sigma(\rho) = (a(\sigma) \cdot \rho + b(\sigma) \bmod p) \bmod 2^\ell.$$

The number p is set to the smallest prime number larger than $2^{\ell+1}$. The functions $a(\sigma)$ and $b(\sigma)$ are the Lehmer [18] random number generators

$$a(\sigma) = \sigma \cdot p_1 \bmod q \qquad b(\sigma) = \sigma \cdot p_2 \bmod q$$

with the prime numbers

$$p_1 = 4645906587823291368 \quad p_2 = 60091810420728157$$

and $q = 2^{63} - 25$ that hold $p_1 \cdot p_2 = 1 \pmod q$. These values were recommended by L'Ecuyer [17] for linear congruent generators. In our implementation, we utilized the fact that

$$a\rho + b = \gamma + i \cdot 2^\ell \bmod p$$
$$\Rightarrow \rho \equiv a^{-1}(\gamma - b + i \cdot 2^\ell) \pmod p$$

with $|i| < p/2^\ell$. This allows a few optimizations when calculating the inverse $F_\sigma^{-1}(\gamma)$, for example, by solving the equation before for (at most) $p/2^\ell$ values of i instead of 2^n using a naïve approach.

C. Implementation details

Our entire implementation measures about 2500 lines of C code and 200 lines of C headers excluding comments and blank lines. Our implementation uses the GNU MP library[2] in version 5.1.2 and a private C framework for memory management, simple object oriented features and lists. Our tool accepts any useful combination of passwords, seeds and digests and finds possible values of the missing parameters. For example: on input of one seed σ and one digest γ the tool will find all passwords ρ with $F_\sigma(\rho) = \gamma$. The tool implements some small optimizations, such as caching the seed calculation results for the 2-universal hash function to accelerate further calculations with the same seed.

D. Performance measurements

The only performance-critical operation a client must support is the selection of seeds σ for given pairs (ρ, γ) of passwords and digests. In order to gauge the overhead of this operation we quantified the number of candidate seeds that must be tested, and we compare the results to the theoretical prediction. We varied the digest length and tested random seeds

[1]http://www.ouah.org/ogay/sha2/, visited on 16.05.2014
[2]http://gmplib.org, visited on 16.05.2014

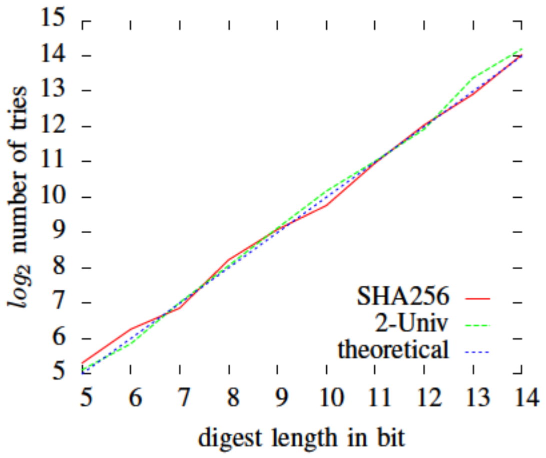

Fig. 6. Plots the number of tries to find a random selected seed with our implementations and the theoretical prediction of the number of tries.

n	s	ℓ	N	best	average	pred.
30	36	6	3	(0.020, 0.001)	(0.016, 0.000)	0.015
*	*	*	4	(0.062, 0.011)	(0.025, 0.002)	*
30	36	6	3	(0.019, 0.005)	(0.016, 0.011)	0.015
*	*	*	4	(0.058, 0.014)	(0.026, 0.006)	*

TABLE II. GUESSING PROBABILITIES FOR EXAMPLE CONFIGURATIONS. ABOVE FOR SHA256, BELOW FOR 2-UNIV. THE MEASUREMENTS ARE IN THE FORMAT (AVERAGE, STDDEV).

of length 30 bits against random digests and random passwords of length 20 bits. We repeated each experiment 100 times and averaged the measurements. Figure 6 shows the results. The line labeled *theoretical* shows the values predicted in Lemma 3. We omit the standard deviation because it was within the predicted ranges. The graph shows that computing seeds for realistic digest lengths is fast on contemporary personal computers. In order to quantify the expected runtime for a concrete example we benchmarked (without optimizations) the number of tests our implementations could perform per second on a MacBook (Model Identifier Macbook4,1) with a 2.1 GHz Intel Core 2 Duo processor and 4 GB of RAM. We found an average of about $1646321 \sim 2^{20.7}$ invocations per second for 2-Univ and about $489687 \sim 2^{18.9}$ invocations per second for SHA256, both with negligible standard deviation. We conclude that finding seeds on commodity personal computers takes fractions of a second.

E. Security measurements

While KYO is fast to use and operate, evaluating its security probability distribution empirically requires computing all pre-images of a hash function invocation within the password space. The password length is bounded from below by the desired security and the number N of pair disclosures that shall be tolerated. Roughly speaking, if digests are ℓ bits long then the length of the password should be approximately $\ell \cdot (N + 1)$. This means that the password space, and hence the necessary computation, quickly exceeds what is feasibly analyzed. Therefore, our future research includes hash functions that lend themselves to more efficient simultaneous inversion for multiple images. One of the prime benefits of KYO is, after all, that it does not require collision resistant hash functions. For the time being, though, we limit ourselves to evaluating the

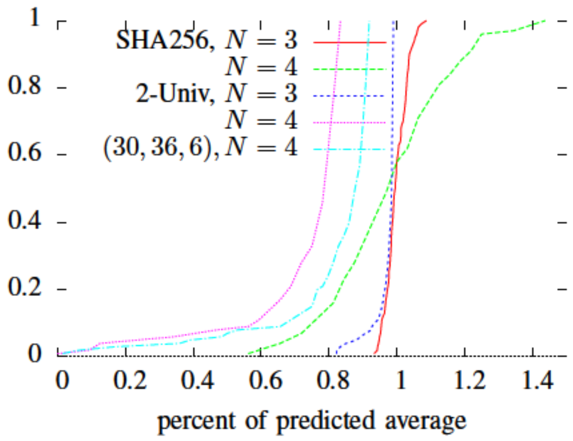

Fig. 7. Cumulative number of passwords in $\cap_i F_{\sigma_i}(\gamma_i)$ in percent of the theoretically predicted average, that is, in y percent of the cases the size was x percent of the predicted average or less.

functions we implemented with artificially small parameters.

We measured the size of the intersections of $F_{\sigma_i}^{-1}(\gamma_i)$ for N randomly chosen values of σ_i and γ_i. These sizes inform our understanding of the *password guessing probability.* Figure 7 shows our results for a configuration of the following lengths: passwords 25 bits, seeds 30 bits and digests 5 bits. We find that both SHA256 and 2-Univ perform on average as we predicted, but especially 2-Univ differs notably for $N = 4$. The last graph for 30 bit passwords, 36 bit seeds and 6 bit digests (named $(30, 36, 6), N = 4$) suggests that larger parameters will yield results closer to the theoretical prediction.

We also measured the *secret guessing probability,* that is, the result of applying Lemma 5 to the aforementioned sets. Table II shows our results for a small number of example configurations. The last column denotes the value predicted by Lemma 5. While the results for $N = 3$ match our predictions pretty closely, the average results for $N = 4$ are slightly worse. We believe the source of these slightly worse results is the small digest length of 6 bit. However, we could not yet verify this hypothesis further due to the high computational complexity. Again, different flavors of hash functions may allow empirical exploration of larger parameters.

VI. Application Notes

By design, KYO is well suited as a password generator to manage web account passwords. As Alice adds server accounts to her address book table, she may update her password length from time to time in order to account for the increasing likelihood that a larger number of servers she uses is compromised along with her client. KYO ensures that the update can happen transparently to server operators. If Alice has a firm understanding of the degree of independence between servers she can use this information to minimize her password length. She shares a password only for mutually independent servers, that is, servers that are not likely breached jointly. Alternatively, Alice may pick her password conservatively in the beginning so that she can keep her password as her number of server accounts grows.

If randomized user names (cf. Florenĉio et al. [10]) are not permissible then Alice may protect herself against denial of service attacks in the following way. In addition to her secret γ, Alice shares a randomly generated token τ. She stores τ in plain text in her address book and sends τ, γ to Bob. Bob only verifies whether Alice's password is correct if the received token τ matches the one he has stored for Alice. It is easy to see that an adversary succeeds in locking out Alice only if he guesses the value τ, the probability of which is negligible in τ. Note that in this case, τ is related to *safety* and not *security.* Hence, Alice only looses denial of service protection (*safety*) if her Address book is disclosed.

Alice may use a similar approach to achieve backwards compatibility with servers that are oblivious to KYO, that is, servers that lock users out only after multiple failed authentication attempts (instead of one failed attempt). As before, Alice generates and stores a token τ in her address book and sends $\tau||\gamma$ for authentication. Note that τ is now related to *security* and Alice sacrifices KYO's protection against address book loss in exchange for achieving backwards compatibility. However, KYO's protection against server-side disclosures remains intact.

Besides the obvious uses of KYO, we originally designed it to help solve a problem that arises in whistleblowing systems. Consider a whistleblower Alice who established contact with a journalist Bob and wishes to send e-mail to the journalist using an anonymous re-mailer. In order to authenticate herself to Bob, Alice encrypts a shared secret along with the message. A typing error would stall her account with Bob and hence it is desirable to check locally that the secret is the correct one. An incorrect secret would then be a giveaway that Alice is in danger. However, it might be desastrous for the whistleblower if the adversary found the secret on her computer because he could use the secret to confirm that she indeed is the whistleblower. With KYO, she can keep her secret on her computer protected with a 4 character password even if the adversary is computationally unbounded.

VII. Related Work

KYO families are an exploration of ideas similar to those of *collisionful hash functions* [3] or *collision rich hashing* proposed previously to secure file checksums against manipulation [19] or to protect key exchange protocols against man-in-the-middle attacks using weak passwords [1]. KYO can be understood as an extension of collisionful hashing to password sharing and password management.

A. Server-side mechanisms

Cappos proposed *PolyPassHash* [6], which is a mechanism meant to mitigate password database leaks from servers. The mechanism keeps users' salted and hashed passwords xor'ed with a per-user *share.* Without knowledge of a share, adversaries cannot brute-force the associated password. Shares are computed using a (k, n)-threshold scheme. The general idea is that once k users have provided their passwords, k shares can be recovered and all other shares can be computed. From this moment on, logins can be verified. In order to allow some level of password verification before the threshold is reached, Cappos suggests that each database entry leak some bits of the salted and hashed passwords.

This bears some resemblance to how KYO uses digests. However, whereas digests are a *safety* mechanism in KYO, they are a *security* mechanism in PolyPassHash. In KYO, a random digest collision results in a fail-safe lockout whereas it leads to a security breach in the case of PolyPassHash. Furthermore, adversaries can use the leaked bits to confirm suspected password re-use, for example, if they have obtained other login credentials of some users in the PolyPassHash database. This in turn allows one to recover shares. KYO, on the other hand, is designed specifically to mitigate the risks of multiple related database leaks (and client leaks).

It is worth noting that PolyPassHash can be broken by bribing at most k users into revealing their passwords after a database leak. This poses no subjective risk to users because they can change their passwords to new ones before revealing their old ones. An even easiest and cheaper attack is to open k accounts at the service before downloading the password database. This means that k passwords are known and all shares can be recovered easily. In the KYO case, password database leaks provide no information on users' passwords unless clients are breached as well, because the seeds are only on clients.

HoneyWords [14] is another mechanism to mitigate password database leaks. The server keeps $k-1$ *decoy* passwords along with each correct one, in random order. When a user logs in, the server queries a trusted *honey checker* service with the user id and obtains the index of the correct password, which is used to verify the user's input. If an adversary obtains the bassword database then he has a $1/k$ chance to choose the right password. If he chooses a decoy then the server locks the associated account. Obviously, the decoys must be chosen so that they are not easily distinguished from the real password. However, adversaries may attempt to exclude decoys by submitting them to *other* services posing as the same user. If the attempt is successful then the user has very likely re-used her password. If the attempt is unsuccessful then the password might have been a decoy and the adversary tries another one.

SAuth [16] also uses decoys to mitigate password database leaks of a service. Decoys are valid alternative passwords, though. Their purpose is not to protect the breached service but to mitigate attacks on other services in cases of password re-use. If two databases are leaked then, again, password re-use is insecure. As in other decoy uses, the challenge is to generate decoys in a fashion that makes them indistinguishable from user-provided passwords. SAuth also requires that users authenticate themselves not only with a valid password but also that they log into a *vouching service* with the password of the vouching service. Both services must be specified beforehand and assurances must be provided that both accounts belong to the same individual. A downside is that users are inconvenienced compared to the status quo because SAuth incurs additional interaction and login delays.

Mechanisms that employ decoys typically require $O(k \cdot n)$ memory where n is the number of passwords and $k-1$ the number of decoys. For large databases, this incurs considerable overhead. If memory is plentiful then passwords can be protected against off-line attacks in a bounded retrieval model [7]. The underlying assumption is that the password database is too large to be leaked in its entirety. KYO, on the other hand, requires $O(n)$ memory, the size of a contemporary password database.

B. Client-side mechanisms

Many password managers, for example, those proposed by Ross et al. [23], [13] and Haldermann et al. [23], [13], apply collision resistant hash functions to various combinations of a user password, user name, site name and a random nonce. A usual assumption is that the hash function is slow enough so that an exhaustive search for the password is infeasible. However, this type of protection requires that passwords are long enough to withstand exhaustive search and that the hash function is regularly updated to reflect advances in algorithms and processor speed. KYO, on the other hand, is designed to be resistant against computationally unbounded adversaries. Assume, for comparison, that calculating a secret takes 1 second on commodity hardware on a single CPU core and further assume that KYO is configured to use 25 bit long random passwords. Since exhaustive searches are trivially parallelized, an eight-core CPU would take at most 48 days to find the password. KYO, on the other hand, remains secure.

Several works propose means to harden password managers against risks such as a client breach. Perhaps the most prominent work is *Kamouflage* due to Bojinov et al. [4]. The authors pointed out that most password manager implementations could be brute-forced. They improved upon the password manager concept by introducing *decoy databases* so that an adversary cannot decide which database holds the correct password for a given service and needs to guess randomly. Failure to guess correctly would eventually cause the attacked account being locked, for example, based on a three-strikes rule. The benefits come at a cost, though. Bojinov et al. recommend 10^5 databases per user as a working configuration. Furthermore, the way in which databases are populated is quite involved because, again, decoys must be generated so that the adversary cannot easily tell them apart from real passwords.

iMobileSitter [12] is a password manager for smartphones. It accepts any master password that is input and, if the master password is false, decrypts its secrets erroneously but plausibly-looking. For example, a four digit PIN is always decrypted into a (different) four digit PIN. In order to hint at incorrect master passwords, it displays an iconic image along with retrieved secrets and expects that the user recognizes that a wrong image is displayed along with a falsely decrypted secret. By virtue of being a password manager rather than a password generator, iMobileSitter is susceptible to known-plaintext or plausible plaintext attacks on the master password. In particular if the secrets have hidden structure, for example, secret strings in *l33t sp34k,* this can be leveraged in a brute-force attack on the master password. This also means that iMobileSitter is not robust against server breaches, which yield known plaintexts.

Additional approaches to hardening secret key encryption use bits of information of a users's personal life in order to encrypt the secret key [8], they use multiple low entropy passwords in order to generate a high entropy password [11] or they leverage secure personal devices for password recovery [20].

On the input error side, there is *password-corrective hashing* [21]. In this approach hash functions map similar passwords to the same output, for example, passwords that differ with respect to character transpositions and substitutions. In contrast to our work, password-corrective hashing does not take password sharing into account. Furthermore, a variety of factors may influence the input errors users make, for example, the keyboard layout and the input method [24]. In particular, soft keyboards on mobile phones were shown to yield different input error rates. We believe it likely that different layouts also have an influence on the types of errors. Password corrective hashing would have to account for all conceivable influences, particularly if passwords are meant to be shared among different devices.

C. Related work summary

In summary, while it may appear that KYO uses elements that have been published prior, none of the cited works and no existing password management solution of which we are aware offers the combination of benefits that KYO offers while KYO suffers from none of the limits and downsides that existing approaches have. Some properties of KYO make it quite unique and an interesting subject for further study. In particular, KYO demonstrates that surprisingly short passwords can offer a high level of security against unbounded attackers while protecting against client and multiple server breaches. Second, KYO demonstrates that fast non-cryptographic hash functions have compelling security applications, particularly hash functions for which pre-images can be enumerated efficiently.

VIII. Conclusion and future work

We have presented KYO, our client-side password generator. KYO is easy to implement, efficient to operate and provides a unique set of benefits that set it apart from the related work of which we are aware. KYO offers short passwords, security against unbounded adversaries and against client and multiple server breaches while being backwards-compatible and flexible at the same time. The feature that we find most compelling is KYO's use of non-cryptographic hash functions. KYO does not rely on collision-resistance and rather benefits from hash functions that enable efficient enumeration and intersection of pre-image sets. We analyzed KYO mathematically and empirically in order to provide evidence of its security, safety and performance. However, its empirical analysis could be pushed to larger security parameters if more beneficial hash functions can be identified. This might also enable a user to test for her choices of passwords what the concrete security is that they provide, rather than relying on average case guarantees, as is common in many areas of cryptography.

Acknowledgments

The first author and the third author were supported by the CONFINE [22] project while doing this research.

References

[1] Anderson, R., and Lomas, T. Fortifying key negotiation schemes with poorly chosen passwords. *Electronics Letters 30*, 13 (Jun 1994), 1040–1041.

[2] Aviv, A. J., Gibson, K., Mossop, E., Blaze, M., and Smith, J. M. Smudge attacks on smartphone touch screens. In *Proc. USENIX Workshop on Offensive Technologies* (2010), pp. 1–7.

[3] Berson, T. A., Lomas, M., and Gong, L. Secure, keyed and collisionful hash functions. Tech. Rep. SRI-CSL-94-08, Stanford Research Institute (Menlo Park, CA US), 1994.

[4] Bojinov, H., Bursztein, E., Boyen, X., and Boneh, D. Kamouflage: Loss-resistant password management. In *Proc. of ESORICS'10* (2010).

[5] Bonneau, J., and Preibusch, S. The password thicket: technical and market failures in human authentication on the web. In *Proc. WEIS* (2010).

[6] Cappos, J. Polypasshash: Protecting passwords in the event of a password file disclosure.

[7] Di Crescenzo, G., Lipton, R., and Walfish, S. Perfectly secure password protocols in the bounded retrieval model. In *Proc. Third Conference on Theory of Cryptography* (2006), vol. 3876 of *LNCS*, Springer-Verlag, pp. 225–244.

[8] Ellison, C., Hall, C., Milbert, R., and Schneier, B. Protecting secret keys with personal entropy. *Future Gener. Comput. Syst. 16*, 4 (Feb. 2000), 311–318.

[9] Florencio, D., and Herley, C. A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web* (New York, NY, USA, 2007), WWW '07, ACM, pp. 657–666.

[10] Florêncio, D., Herley, C., and Coskun, B. Do strong web passwords accomplish anything? In *Proceedings of the 2Nd USENIX Workshop on Hot Topics in Security* (Berkeley, CA, USA, 2007), HOTSEC'07, USENIX Association, pp. 10:1–10:6.

[11] Frykholm, N., and Juels, A. Error-tolerant password recovery. In *Proceedings of the 8th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2001), CCS '01, ACM, pp. 1–9.

[12] GmbH, M. imobilesitter —mobilesitter for iphone. http://www.mobilesitter.de/downloads/mobilesitter-kes-special-mobile-security.pdf.

[13] Halderman, J. A., Waters, B., and Felten, E. W. A convenient method for securely managing passwords. In *Proceedings of the 14th International Conference on World Wide Web* (New York, NY, USA, 2005), WWW '05, ACM, pp. 471–479.

[14] Juels, A., and Rivest, R. L. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security* (New York, NY, USA, 2013), CCS '13, ACM, pp. 145–160.

[15] Kaliski, B. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898 (Informational), September 2000.

[16] Kontaxis, G., Athanasopoulos, E., Portokalidis, G., and Keromytis, A. D. Sauth: Protecting user accounts from password database leaks. In *Proc. CCS* (2013), pp. 187–198.

[17] L'Ecuyer, P. Tables of linear congruential generators of different sizes and good lattice structure. *Math. Comput. 68*, 225 (Jan. 1999), 249–260.

[18] Lehmer, D. H. Mathematical methods in large-scale computing units. In *Proceedings 2nd Symposium on Large-Scale Digital Computing Machinery* (Cambridge, MA, 1951), Harvard University Press, pp. 141–146.

[19] Lomas, M., and Christianson, B. Remote booting in a hostile world: to whom am i speaking? [computer security]. *Computer 28*, 1 (Jan 1995), 50–54.

[20] Mannan, M., Barrera, D., Brown, C. D., Lie, D., and van Oorschot, P. C. Mercury: Recovering forgotten passwords using personal devices. In *Proceedings of the 15th International Conference on Financial Cryptography and Data Security* (Berlin, Heidelberg, 2012), FC'11, Springer-Verlag, pp. 315–330.

[21] Mehler, A., and Skiena, S. Improving usability through password-corrective hashing. In *Proceedings of the 13th International Conference on String Processing and Information Retrieval* (Berlin, Heidelberg, 2006), SPIRE'06, Springer-Verlag, pp. 193–204.

[22] CONFINE. Community networks testbed for the future internet. http://confine-project.eu/, June 2013.

[23] Ross, B., Jackson, C., Miyake, N., Boneh, D., and Mitchell, J. C. Stronger password authentication using browser extensions. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14* (Berkeley, CA, USA, 2005), SSYM'05, USENIX Association, pp. 2–2.

[24] Schaub, F., Deyhle, R., and Weber, M. Password entry usability and shoulder surfing susceptibility on different smartphone platforms. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia* (New York, NY, USA, 2012), MUM '12, ACM, pp. 13:1–13:10.

[25] Wegman, M. N., and Carter, J. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences 22*, 3 (1981), 265 – 279.

Appendix

Before we start we recapitulate a well-known lemma that we use throughout our proofs.

Lemma A1 (Number of functions)**.** *Let X_1, X_2 be non-empty and disjoint sets and Y_1, Y_2 non-empty sets. Set $X := \cup_i X_i$, $Y := \cup_i Y_i$, and*

$$F := \{f : X \to Y \mid \wedge_i \ \forall x \in X_i : f(x) \in Y_i\}.$$

Then it is $\mid F \mid = \mid Y_1 \mid^{|X_1|} \cdot \mid Y_2 \mid^{|X_2|}$.

KYO properties:

Lemma A2. *Let σ be a seed and γ a digest. Then the size of $F_\sigma^{-1}(\gamma)$ is binomial distributed with average value $2^{n-\ell}$.*

Proof: For a password ρ, what is the probability that it is mapped to γ? There are in total $(2^\ell)^{2^n-1}$ functions that do map a given ρ to a given γ. So the probability to select such a function is

$$p := \frac{(2^\ell)^{2^n-1}}{(2^\ell)^{2^n}} = 2^{-\ell}.$$

We now get for the number of passwords that map to γ_i

$$\Pr[\mid F_\sigma^{-1}(\gamma) \mid = x] = \binom{N}{x} p^x (1-p)^{N-x}.$$

So $F_\sigma^{-1}(\gamma)$ is binomial distributed with expected value $2^n \cdot 2^{-N \cdot \ell} = 2^{n-\ell}$. ■

Lemma 1 (Password creation)**.** *If the password table PW already contains N passwords, the number of random tries to find the $(N+1)$-th password is geometric distributed with average value $(1 - N \cdot 2^{-\ell})^{-1}$.*

Proof: Given $(\sigma_1, \gamma_1), \ldots, (\sigma_N, \gamma_N)$, what is the size of $\cup_i F_{\sigma_i}^{-1}(\gamma_i)$? We know from Lemma A2 that $\mid F_{\sigma_i}^{-1}(\gamma_i) \mid$ has average value $2^{-\ell}$. So we get on average

$$\mid \cup_{i=1}^N F_{\sigma_i}^{-1}(\gamma_i) \mid = N \cdot 2^{n-\ell}.$$

So the probability to successfully pick a suiting password at random is

$$p := 1 - \frac{N \cdot 2^{n-\ell}}{2^n} = 1 - N \cdot 2^{-\ell}.$$

The number of random tries to find one password is then given by the geometric distribution with probability p, and its expected value is $1/p$. ■

Lemma A3. *Given a password ρ and secret γ, the number of random tries to find a seed σ that satisfies $F_{\sigma_i}(\rho) = \gamma$ is geometric distributed with average value 2^ℓ.*

Proof: The probability to pick such a function in one try is $2^{-\ell}$. Let X be a random variable that denotes the number of tries until one is found. Then X is geometric distributed:

$$\Pr[X \le x] = 1 - \left(1 - 1/2^\ell\right)^x.$$

The expected value of X is $2^{-\ell}$, so one must test 2^ℓ different seeds on average to find one such seed. ■

Lemma 2 (Seed selection 1)**.** *For given $N-1$ passwords, a randomly chosen seed σ fulfils requirement 2 with probability $(1-2^{-\ell})^{N-1}$.*

Proof: We count the number of functions that do not map the passwords $\rho_1,\dots,\rho_{N-1}$ to an arbitrary but fixed digest. From lemma A1 follows that there are

$$(2^\ell - 1)^{N-1} \cdot (2^\ell)^{2^n-(N-1)}$$

of them. Since there are $(2^\ell)^{2^n}$ functions in total, we get the probability that a randomly chosen function fulfils requirement 2

$$\begin{aligned}(2^\ell - 1)^{N-1} \cdot (2^\ell)^{-(N-1)} &= \frac{(2^\ell-1)^{N-1}}{(2^\ell)^{N-1}} \\ &= (1-2^{-\ell})^{N-1}.\end{aligned}$$

■

Lemma 3 (Seed selection)**.** *Given a password ρ and shared secrets $\gamma_1,\dots,\gamma_N$, the number of random tries to find seeds $\sigma_1,\dots,\sigma_N$ that satisfy $F_{\sigma_i}(\rho)=\gamma_i$ is geometric distributed with average value $N\cdot 2^\ell$.*

Proof: Follows by N repetitions from Lemma A3 . ■

Safety:

Lemma 4 (Safety)**.** *Let N be the number of pairwise distinct wrongly entered passwords and let N be small compared to 2^s. Then the safety of the KYO scheme is binomial distributed with average value*

$$1-(1-2^{-\ell})^N.$$

Proof: Let γ be the stored digest. There are $(2^\ell-1)^N\cdot(2^\ell)^{2^n-N}$ functions that do not map the passwords ρ_i to the digest γ. We now find, that

$$\begin{aligned}&\Pr[\{\sigma\in\Omega : \vee_{i=1}^N F_\sigma(\rho_i)=\rho\} = \\ &=1-\Pr[\{\sigma \mid \wedge_{i=1}^N F_\sigma(\rho_i)\neq\rho\} \\ &=1-\frac{\mid\{\sigma \mid \forall\rho\in\{\rho_1,\dots,\rho_N\} : F_\sigma(\rho)\neq\rho\}\mid}{\mid S\mid} \\ &=1-\frac{(2^\ell-1)^N\cdot(2^\ell)^{2^n-N}}{(2^\ell)^{2^n}} = 1-(2^\ell-1)^N\cdot 2^{-\ell N} \\ &=1-(1-2^{-\ell})^N.\end{aligned}$$

If N is small compared to 2^s, we can approximate this experiment by drawing from an urn with replacement. The probability is then binomial distributed with the average value above. ■

Security:

Lemma 5 (Guessing secrets)**.** *Let C be a set of equally likely candidate passwords and $c := |C|$. We define $M := F_\sigma(C)$. Then the probability*

$$\Pr[\Gamma=\gamma \mid \Sigma=\sigma\wedge\rho\in C]$$

is binomial distributed with the average value

$$\left(2^\ell\cdot(1-(1-2^{-\ell})^c)\right)^{-1}$$

for every $\gamma\in M$.

Proof: In this proof, we consider F to be a random function. Note that this does not contradict our definition of KYO families, but instead provides an easier proof. We model this situation by drawing balls from an urn with 2^ℓ different colored balls for c times.

For each secret γ, the probability that it is mapped onto by at least one password in C is $1-(1-2^{-\ell})^c$. So the size of M is binomial distributed with average value $2^\ell\cdot(1-(1-2^{-\ell})^c)$.

Let X denote a random variable that gives the number of passwords that map to a given digest. Since F is a random function, X too will be binomial distributed with average value $c\cdot 2^{-\ell}$. So on average, each digest $\gamma\in M$ will be equally likely with probability $1/c$. ■

Lemma 6. *The size of M is binomial distributed with average value*

$$2^{n-N\cdot\ell}.$$

Proof: For a password ρ, what is the probability that it is mapped to γ_i? There are in total $(2^\ell)^{2^n-1}$ functions that do map a given ρ to a given γ_i. If the σ_i are selected uniformly at random and independent from each other, we get

$$p := \left(\frac{(2^\ell)^{2^n-1}}{(2^\ell)^{2^n}}\right)^N = 2^{-N\cdot l}$$

as the probability that such a function was selected each time. We get for the number of passwords M that map to γ_i

$$\Pr[M=m] = \binom{N}{m} p^m(1-p)^{N-m},$$

i.e., M is binomial distributed and it's expected value is

$$2^n\cdot 2^{-N\cdot\ell} = 2^{n-N\cdot\ell}. \tag{1}$$

■

Theorem 1 (Guessing passwords)**.** *It is*

$$\Pr[R=\rho \mid \wedge_{i=1}^N(\Sigma_i=\sigma_i\wedge\Gamma_i=\gamma_i)] = \begin{cases}\frac{1}{|M|} & \rho\in M \\ 0 & else.\end{cases}$$

with the size of M being binomial distributed with average value 2^{n-Nl}.

Proof: For any $\rho_i\in M$, we get using Bayes' theorem

$$\begin{aligned}&\Pr[R=\rho \mid \wedge_{i=1}^N(\Sigma_i=\sigma_i\wedge\Gamma_i=\gamma_i)] = \\ &=\frac{\Pr[\wedge_{i=1}^N(\Sigma_i=\sigma_i\wedge\Gamma_i=\gamma_i)\mid R=\rho_i]\cdot\Pr[R=\rho]}{\sum_{\rho_j\in M}\Pr[\wedge_{i=1}^N(\Sigma_i=\sigma_i\wedge\Gamma_i=\gamma_i)\mid R=\rho_j]\cdot\Pr[R=\rho_j]},\end{aligned}$$

which simplifies to

$$\frac{\Pr[\Sigma_i=\sigma_i\wedge\Gamma_i=\gamma_i \mid R=\rho_i]}{\sum_{\rho_j\in M}\Pr[\Sigma_i=\sigma_i\wedge\Gamma_i=\gamma_i \mid R=\rho_j]} = \frac{1}{\mid M\mid}, \tag{2}$$

since all passwords are initially equally likely and the terms

$$\Pr[\Sigma_i=\sigma_i\wedge\Gamma_i=\gamma_i \mid R=\rho_i]$$

are all equal, given that $(F_\sigma)_{\sigma\in S}$ covers all functions mapping P to L. The result follows now from Lemma 6. ■