Cue-Pin-Select, a Secure Mental Password Manager

Anonymous authors
Redacted institutes

***Abstract*—People struggle to invent safe passwords for many of their typical online activities, leading to a variety of security problems when they use overly simple passwords or reuse them multiple times with minor modifications. Having different passwords for each service generally requires password managers or memorable (but weak) passwords, introducing other vulnerabilities [1], [2]. Recent research [3], [4] has offered multiple alternatives but those require either rote memorisation [5] or computation on a physical device [6], [7]. This paper describes a secure and usable solution to this problem that requires no assistance from any physical device.**

We present the Cue-Pin-Select password family scheme that requires little memorisation and allows users to create and retrieve passwords easily. It uses our natural cognitive abilities to be durable, adaptable to different password requirements, and resistant to attacks, including ones involving plain-text knowledge of some passwords from the family. We include a theoretical analysis of its security according to multiple attack models. Finally, we show the promising results of a small-scale user study that put participants in real-life conditions for multiple days.

***Index Terms*—Usable-security; Password schemes; Authentication;**

I. Introduction

As has been observed repeatedly [8]–[10], the number of passwords is ever-increasing, and having different passwords for each service generally requires password managers or memorable (but weak) passwords, introducing other vulnerabilities [1], [2], [11]. This is often the result of an unconscious trade-off between security and usability, sometimes leading to cognitive dissonance [10]: although users know they are vulnerable, they do not take actions to remedy this [12], [13]. A wide variety of factors affect this choice, among which mainly stand effort, lack of information about alternatives and lack of perceived usefulness [14]. Inadequate mental models of security also play a role [12], [15]–[17].

This isn't only the users' fault: well-meaning but counterproductive constraints (such as mixed-case, numbers and symbols) have been mostly detrimental [16], [18], [19]. They not only pushed users to have weak passwords — focusing their efforts on satisfying or bypassing the constraints instead of making good passwords — but also forced them to create passwords in an ad-hoc way, preventing them from following habits which improve memorisation. Those new passwords become not only weak but forgettable, and lead to frequent resets of the rarely used passwords. The traditional solutions for users have been to write down their passwords [20], use the same few passwords for everything [9], or use password managers, constituting a single point of failure from which an adversary can completely steal an identity [21]. Alternatives are being developed but biometric authentication is still suffering from serious flaws [22]–[24], and so do password managers as they increase reliance on either specific — and potentially unreliable — web services [21], or on one's own devices which can stop working or get stolen.

A different potential solution is to create a set of non-independent passwords, related by a common pattern. As humans are natural pattern seekers, many intuitive ways have been devised to avoid password re-use without incurring too high a mental cost [25]. Those schemes which create new passwords automatically can be arbitrarily simple or complex, going from very small variations at the end of a word to word association schemes [5], [26]. They alas tend to have security barely above password re-use, as users may produce families of related passwords which an adversary can easily infer if they learn some examples (such as Mypassword1! Mypassword2...). Finding a good method to remember large sets of passwords at little cost to the user would then be a boon.

II. Previous work and contributions

There have been a few recent attempts at password managers that are not device-reliant, but they generally require a large amount of rote memorisation [3], [5] or computation on a physical device [6], [7]. Efforts have mostly been led by Manuel Blum and his co-authors N. Hopper, J. Blocki, A. Datta and S. Vempala, with six papers on the subject, including five in the past five years [4], [5], [7], [27], [28]. In them, they have framed, formalised and brought forward many important issues in mental passwords managers and password schemes, on both the security and usability fronts. They mention five criteria that schemes should satisfy:

- *Analysable*, meaning that the schema should be a well-defined and deterministic algorithm.
- *Publishable*, corresponding to Kerckhoffs's law.
- *Secure*, typically resisting both online and offline attacks from an adversary with superior computing power.
- *Self-rehearsing*, such that the process of using the scheme regularly enough is sufficient to remember it.
- *Humanly usable*, such that an average human can learn and use the system at no great cost.

They have also proposed a series of algorithms, generally based on challenge systems, where the authenticating system sends some information to the user, to which they must respond accordingly. The challenges are combined with computational primitives that the users are supposed to execute. For example, their DS1 protocol requires a user to memorise a random letter-to-digit map as the seed to their password creation, while WS1 asks users to memorise both letters and

words about a topic [5]. They also describe LP1 that requires users to memorise a random letter permutation. These different kinds of memory tasks are all valid, but memorising random mappings is difficult, and may take time and a large amount of effort.

One of the most interesting systems they created, for which they prove strong security bounds, requires the user to remember a mapping between digits and a set of 14 images [7]. Once they have the mapping, the challenge system shows the images on the screen in a random order, and the user has to compute

$$x_{13} + x_{12} + x_{(x_{10}+x_{11} \mod 10)} \mod 10$$

where x_i is the number corresponding to the i-th image in the list shown to the user. Although this is complex enough, authenticating using this kind of system requires at least 3-5 different challenges. The original paper mentions that the main author managed — with some training and solid natural mathematical abilities — to reliably compute the function for a single challenge in 7.5 seconds, getting to an adequate security level in at least 30s. This performance is however not a priori representative of the average user.

In spite of their excellent work and prescient points about what would be usable password creation and retrieval, people today still don't know which method to use. The algorithms they propose are already quite expensive time-wise, but more importantly, they are not directly applicable by end-users as they generally rely on service providers changing their security systems. Moreover, it seems that the criteria they created might not be enough. Users are still left with no good algorithm to easily create secure families of passwords.

Contributions: This paper introduces three new criteria and proposes and characterises a specific usable scheme (though certainly not the only one possible) that allows the easy creation, memory, usage, and retrieval of a password, while remaining secure against a large variety of attacks. The criteria are as follows:

- *Agent-independence*, meaning that the user should not require any outside help, be it from their own computing devices, an untrusted calculator or phone, or even pen and paper.
- *Scalability*, an extension of their *publishability* criterion, with two constraints: the security of the scheme should not strongly diminish with either increased popularity or increased use by a single user with many different passwords.
- *Adaptability*, the final and most important criterion, permitting the user to always be able to use the system no matter the idiosyncratic constraints they face.

This last criterion is crucial as otherwise the user is faced with two possibilities: creating a new password in an ad-hoc way, which might be insecure and which they will probably forget, or change the algorithm used at some point. Moreover, remembering many algorithms, their associated secret information and where they were used seems even harder than remembering many passwords.

The system we propose, called *Cue-Pin-Select*, is a password generation scheme for 12+ characters that is provably strong and adaptable to the requirements of today's systems. Designed to profit from our natural linguistic abilities, it performs well on constraints of usable memory and learning, while fulfilling strong security constraints. The system relies on selecting a cue from or for a service onto which the user might log (such as part of its name), and applying a PIN to create an index into a common six word phrase. The intent behind its creation was to show a system that cannot be reasonably hacked even if the adversary knows some of the plain-texts, while also making it human computable without command-line tools or too much work. We can learn six words and a PIN much more easily than the random mappings proposed and they are directly usable without any specific knowledge or ability. A preliminary user study showed how all alpha-testers with only a few minutes of training were able to produce a new secure retrievable password in well under a minute and improved their times consistently in 19 trials over 4 days. Even with this limited training, the fastest users quickly achieved performance similar or better than that shown in [7]. Although both systems have performance that are relatively similar, CPS has the added advantage that the algorithm can be used as is, without requiring a complete redesign of server-side authentication systems.

In general considerations on password schemes, we will suppose that each scheme is composed of three kinds of information:

- Some user-only information which can include the initial seed for random data, or any piece of information that should absolutely not be spread. If an adversary obtains it, this information can potentially allow them to recreate all the user's passwords.
- The passwords themselves, from which it should be difficult to find the user-only information.
- An environmental cue, a piece of information (in our case four characters), or the scheme used, from which a user can (re)create a password if they have the user-only information. Those can be guessable and should offer little direct information on the rest by themselves.

The rest of the paper follows the following structure. We start by explaining the proposed scheme and analyse its resistance to the main types of attacks. We then look at it from a usability standpoint and show the promising results of a small-scale user study that put participants in real-life conditions for multiple days. Finally, we introduce multiple variants and explain the design choices leading to Cue-Pin-Select

III. The Cue-Pin-Select Scheme

Cue-Pin-Select uses four different pieces of information. A user starts with one long high-entropy passphrase that is highly memorable despite its length, and a 4-digit PIN. The process uses an algorithm that is easy to remember and implement, and finally, for each service where a user needs a password, they need to choose a small four-letter string called the *cue*.

A. Passphrase

The main secret data has two components. The first is a passphrase of at least 6 English words[1], and the second is a 4-digit PIN of the kind that people are accustomed to associating with bank cards. Both the passphrase and the PIN are common to all passwords in the scheme. To generate the passphrase, the user is supplied with a random sequence of 6 words, to which they can add connecting words.

In the usability test, participants were encouraged to use an online random word generator, but more sophisticated methods could also be used to create a long and memorable passphrase. We use the methods shown in in [29] for our security analyses, considering words taken quasi-randomly from a dictionary of the 87 691 most frequent correct English words, using Peter Norvig's list of most frequent ngrams [30], as it has been shown that this method can provide memorable passphrases with 97% of the entropy of uniform sampling among the same dictionary. The 4-digit number is also randomly generated (for example by rolling dices).

B. Password generation algorithm

Each time the user needs a password for a new service, they need only apply the Cue-Pin-Select algorithm.

The algorithm makes a 12-character password in 12 steps. Let's suppose a user has created the passphrase *parallel major domain disastrous divergent waterways* and that their PIN is *6908*. Say they are making the password for their Amazon account. They start by coming up with a 'cue': a 4-character string corresponding to this service, say *amzn*. This cue will then be used to extract password parts from the passphrase.

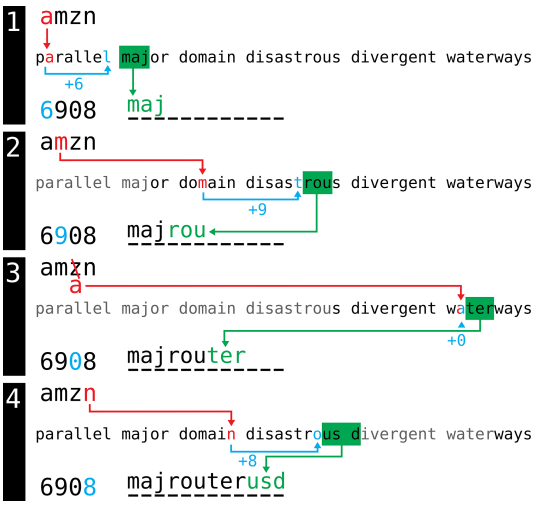

Fig. 1. Running the four phases of Cue-Pin-Select using AMZN as a cue and *parallel major domain disastrous divergent waterways* as a passphrase.

Figure 1 shows this process, where each operation has a colour, creating the password `majrouterusd`, using Algorithm 1. Once they have chosen (or remembered) their cue, they proceed as follows:

Algorithm 1: Cue-Pin-Select

Data: Passphrase $PHRASE$ of at least 6 random words
PIN of 4 random digits
service name $NAME$
Result: String S of 12 characters

begin
  From $NAME$, create string CUE of four characters
  /* User-chosen, which should be easy to remember */
  $LEN \longleftarrow$ Length($PHRASE$)
  $INDEX \longleftarrow 0$
  $S \longleftarrow []$
  for $i = 0\ ;\ i < 4\ ;\ i++$ **do**
    $LETTER \longleftarrow CUE[i]$
    while $LETTER \notin PHRASE$ **do**
      $LETTER \longleftarrow$ letter following $LETTER$ in the alphabet
    $INDEX \longleftarrow$ index of next occurrence of $LETTER$ *in* $PHRASE$ after $INDEX$
    /* Or wrap around to the first occurrence if the end of $PHRASE$ is reached */
    $INDEX \longleftarrow (INDEX + PIN[i] + 3) \mod LEN$
    $S \longleftarrow$ Concatenate $(S, PHRASE[INDEX - 2, INDEX - 1, INDEX])$
    /* All the indices are modulo LEN */
  Print S

1) They look for the first letter of their cue, `a`, in the passphrase. In our example, this would be the first `a` found in the word `parallel`. They then step though the letters indicated by the first number of their PIN, in this case 6. This would be the last `l` of `parallel`. They add the next three letters in their passphrase to their password, `maj`.
2) They look for the next letter, `m` from where they left off. This leads them to the `m` of `domain`. They skip 9 letters, getting to the `t` of `disastrous`, and add the next three letters, `rou` to their password.
3) They look for the next letter, `z`, but can't find it. As the letter in the cue isn't in the passphrase, they look for the next letter in the alphabet: `z` is then replaced by `a`, and they continue where they left off. The next `a` is the first one in `waterways`, and as the third number in their PIN is 0, they take the next three letters, `ter`.
4) For the last step, they have to look for an `n`, but reach the end of the sentence, they then continue from the start, get the `n` of `domain`, skip 8 letters and end up with `usd`.
5) They are then left with their password: `majrouterusd`.

C. Finding forgotten passwords

As the procedure is deterministic — for a given passphrase — the only variability comes from the cue. In case they forget their original cue, the user should be able to find it within a

[1]The choice of English was for the entropy analyses and the user study, but the scheme is adaptable to any language that is written in an alphabetic system.

few tries, from which they can derive the whole password. However, there is another simpler option: the cue and the PIN could hypothetically both be written down by the user, as the security analyses don't assume that they are secret. This way, only the passphrase stays secret, and it is the most frequently rehearsed bit.

The analyses in the coming section pertain to the model shown here. Variants to the algorithm can be introduced for making new passwords or responding to various password requirements. Some representative variants will be studied after the analysis below.

IV. Security analysis

As a combinatorial analysis of all combinations of words from the dictionary with the algorithm would be intractable and highly dependent on specific properties of our dataset, analyses here rely on Monte-Carlo models. The entropies are computed exactly from the k-grams index, the list of k letter sequences present in sentences made from the dictionary.

A. Preliminary considerations

One of the main assumptions used in the following analyses is that the distribution of three-letter trigrams composing each password is very close to the distribution of a random trigram taken in a random passphrase. The PIN is an essential part of the randomisation mechanism. It is important because simply reading a sequence of characters in words when reading the cue letter `q` without the PIN step would give trigrams like `qu` where `q` is followed by `u` in 1248 out of 1266 cases, with only 2.7 bits of entropy. An `o`, however, would give 10.4 bits as it reveals little information on the following characters.

Distribution uniformity is nearly achieved as the number of characters stepped over each time through the passphrase is random enough that the probabilities of landing on any letter of a given word are quasi-uniform. The simulation shown in Figure 2 presents four curves[2] representing the probability distribution for the number of letters stepped over (one for each letter in the 4-character cue). Since it is much bigger in expectation than an average word length of 9, the probabilities of landing on the n-th letter of a word are then close enough to uniform along n to provide no real advantage to an adversary.

The length of the passphrase itself follows a Bell-like distribution (being a product of distributions that are themselves Bell-like). It has 99% probability of being between 33 and 65 characters long, centred around 48 and has a large variance. As a consequence, with high probability, the second trigram comes later than the first in the passphrase. However, thanks to the large variance in the probability function, the probabilities of the second trigram preceding or following the third trigram are not too far apart, and the same can be said for all the other pairs.

[2]The shape of those curves might seem to follow Zipf's law [31], with the number of letters covered being inversely proportional to the letter's rank frequency — to which an offset has been added because of the random PIN. However, in such a case the maximum would be reached with fewer than 10 letters stepped over.

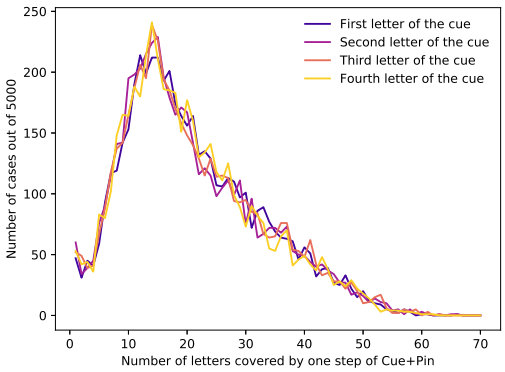

Fig. 2. Distribution of the number of characters covered in one step of Cue+Pin, obtained by simulation on 5 000 (passphrase/cue) pairs, for passphrases of average length 48.

B. Scalability

1) Publishability: The scalability of a scheme corresponds to two different notions. First, it should be scalable in number of users. Any person who uses an obscure but adequately complex scheme will be protected by a lack of specialised attacks targeting it. This is not true for a scheme used by millions of people. As such, all the threat models should assume Kerckhoffs's principle that only the key (cue, PIN, and service code) are private, the algorithm being public. This corresponds to [5]'s notion of *publishable*. In our situation, there could actually be a small positive impact of large-scale implementation, in that people using it in a variety of languages reduces the possibility of statistical and dictionary attacks, marginally increasing the general level of security (as opposed to only American English users).

2) Creating many passwords: The second type of scalability corresponds to the number of passwords used by a single user; frequently using a scheme should not make it more vulnerable (besides the higher risk of multiple plain-text attacks). For a given attacker with specific computing resources, knowing some plain-text passwords, and other information, the probability of uncovering passwords should only marginally increase with the number of passwords created by the user through the scheme. A simple unscalable example would be a system that solely depends on a passphrase composed of four sections, where the user randomly selects two sections each time they need a password. If they use this scheme less than 3 times, assuming each section has sufficient entropy, passwords don't reveal each other. After 7 uses, however, some of the passwords will be repeated. On the other hand, while having a completely new password for each new service is infinitely scalable, as described above, it will require some way of remembering the passwords, which introduces vulnerability.

For Cue-Pin-Select, it is enough to show that all the passwords generated will be different from each other — unless the server itself is malicious, which is treated further down. It's clear that it should be the case in general, when the user has different cues (in particular, ones that don't have the same first three letters). However, a simulation where each passphrase

generates 20 passwords demonstrated that the average distance between two passwords is close to what would be expected from two random strings (at most a few letters being shared). The edit distance between the two closest passwords generated was also calculated (corresponding to the risk of having one other password stolen when the worst password is stolen). This showed that even in this worst situation, in more than 99% of cases an adversary would have to change at least three letters (a quarter of the password, although they don't know which one, corresponding to 15 bits of security[3]), assuming they already possess one of the two closest passwords.

C. Brute-force and dictionary attacks

1) Attacking the password: Current entropy recommendations against brute-force attacks vary from 29 bits to 128 bits of security, depending on the attack model [32]. One common recommendation proposes 36 bits of security on any given password for web services; such a password would require 1 000 tries per second for one year to break. Assuming the attacker uses online servers to distribute the attack (and avoid rate-limiting), in 2019 this would require more than $1000 per password, even with strong economies of scale [33]. This is consistent with NIST recommendations as long as a key derivation function is used [34], which limits the amount of offline computation.

In our case, assuming the adversary knows the scheme used, a smarter attack would be to guess which trigram is used in each position. However, an analysis of the distribution of trigrams in the dictionary shows that each trigram adds around 13 bits of entropy. To get this number, we computed explicit probabilities for each potential trigram in English using the SOWPODS dictionary — not only within words but also trigrams crossing over words. We also assumed one additional hypothesis: that the start of the trigram in the word is uniformly distributed. This is not entirely accurate, as it depends on the letter chosen in the Cue phase, but the Pin phase adds enough uncertainty to make it quite uniform (as was tested on random sentences and PINs. Explicit computation on trigrams gave the amount of entropy per trigram — within our dictionary — depending on the type of trigram, which can be either within a word or across two words. Computing them showed that the entropy is highest for trigrams composed of the last letter of a word and the first two letters of the next, with 13.95 bits. The lowest was for trigrams composed of the last two letters of a word and the first of the next, with 11.42 bits, and trigrams within single words had 13.17 bits of entropy. As the latter are by far the most frequent type of trigrams, it is reasonable to assume that each password has around 52 bits of entropy. This is close to the optimal performance of uniform alphabetic passwords of length 12, which have 56.41 bits of entropy.

[3]Although 15 bits of security might seem low, it stills corresponds to more than 10 000 login attempts, assuming that the adversary is lucky and already knows the closest password and not just a random password.

2) Attacking the passphrase: The passphrase is much more valuable than any of the passwords; however, it also has much higher security. Indeed, the six mandatory words are uniformly distributed among a dictionary of 87 691, leading to a raw entropy above 98 bits. Adding the PIN gives 111 bits of entropy, way more than any user could reasonably use, even against distributed attacks. Two factors reduce this value: the user chooses the order of the passphrase, which can reduce entropy by 3-7 bits depending on the model, and they can also redraw random words a few times if they don't like the first ones (removing one or two bits). This small cost is partially compensated by the fact that they can use auxiliary words. Overall, assuming the worst case and finicky users, the passphrase and PIN should still give at least 102 bits of entropy. Against dumb brute-force attacks, it would have more than 210 bits of entropy, confirming the problem with using raw entropy without specifying the adversarial model.

3) Resistance to plain-text attacks: Plain-text attacks are one of the main vulnerabilities found in most user behaviours today, generally stemming from password reuse. This is also where typical methods as described by LifeHacker fail [26]. Assume that, with the drop in computing costs, the adversary tries not just the exact password they have access to but also simple variants of it. The remaining entropy should stay high, even assuming that the adversary knows both the method and at least one plain-text password [20].

The scheme was designed to provide high security even in the event that one (or even a few) of the passwords are compromised, which can happen independently of the user's best practices. As said earlier, trying to guess one password from another in Cue-Pin-Select is a hard problem in the general case, as the edit distance is great (only marginally lower than the edit distance between the password and a random string). The easiest way of attack then seems to go through the derivation of the passphrase from a password obtained by the attacker.

a) Plain-text attacks: To analyse the security of the passphrase from plain-text attacks, suppose that the adversary knows not only the plain-text but also the length of the passphrase and the position of the plain-text inside the passphrase. This gives way more information to the adversary than is realistic, due to the variation in passphrase length discussed above. Even in such a case, we can show that it is hard to find the passphrase from a single password.

10^4 random (passphrase/cue) couples were computed to get in each case a passphrase where only certain characters were revealed. Dynamic programming was then used to compute the number of passphrases that used exactly 6 words, compatible with the revealed letters and had the right length. This gave the number of potential combinations in each case, which is shown in logscale (hence corresponding to the number of bits of entropy) in the top curve of Figure 3.

This shows that Cue-Pin-Select can guarantee a minimum of 40 bits of entropy in case of a plain-text attack, with an average of 54 bits (the standard deviation is 6 bits), and a maximum of 79 bits.

The curves for the remaining entropy, when the lucky adversary has access not only to the length and positions of revealed letters, but also to either two or three passwords, are shown on the bottom curve of Figure 3 (5×10^3 runs each).

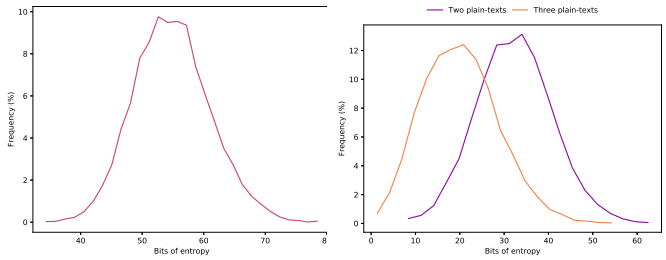

Fig. 3. Bits of entropy left on a passphrase when a plain-text and the position of its letters are revealed. The top figure corresponds to a single plain-text, while the bottom one features the curves for 2 and 3 plain-texts. Obtained by simulating 10 000 random (passphrase/cue) couples – for the top figure — and 5 000 (passphrase/cue 1/cue 2/cue 3) tuples for the bottom figure. The bits of entropy come from the exact number of possibilities remaining using the passphrase length and the revealed passwords (computed from the cues) as constraints.

In those two curves, the average number of bits of entropy left is respectively 32 and 20 bits. However, a large standard deviation (around 9 bits in both cases) and a variability in passphrase length means that in degenerate cases (which happened once in each group of 5 000 simulations of the adversary having several revealed passwords, the length and position of the revealed letters), a double plain-text lead to only 7 bits of entropy, and a triple plain-text completely revealed a passphrase. This is to be expected as this would reveal up to 36 characters, and close to 2% of passphrases are smaller than that. As some of the plain-texts can also give redundant information, the maximal entropies left were 64 and 56 bits.

The guaranteed high level of entropy against at least two plain-text attacks means that users should be secure if they maintain good security hygiene and change their passphrase when they think they have been compromised. Even adversaries with decent computational means and knowledge of the system used should have no real chance of cracking their passphrase.

4) Resistance to side-channel and other attacks: Schemes have been proposed that are resistant to brute-force while requiring little effort on the user's part, but require some computation, or storing of information on a trusted device. This can be as simple as a persistent physical memory, corresponding to writing passwords in a notebook, using a password manager on a potentially vulnerable computer (e.g. to keyloggers), or, as in [4], [7], requiring semi-secure computation in the form of challenges from a computer. It can also include hybrid methods such as the password card [35], [36], in which obtaining the card does not give complete access but reduces the entropy of the owner's passwords to about 10 bits each.

As Cue-Pin-Select can stay entirely in the user's mind, it should be entirely secure against side-channel attacks, as the only links between the passwords are entirely immaterial. Those passwords are the only information available no matter the adversary's means, so the security of the scheme corresponds exactly to the security shown above.

This, however, ignores three possibilities. The first is that someone could know the user well enough that they could guess the user's choices. While other schemes that rely on mental association are also potentially vulnerable to someone who knows the user extremely well (even more so if they also have access to computational power to get through the last bits of entropy), Cue-Pin-Select does not. This is why the words of the passphrase should be generated randomly, in a way that doesn't depend on the user's typical choices.

The second possibility is more down-to-earth: some users might write down their passphrase to help them create their first few passwords, or to create a new one. As long as they destroy this physical (or digital, if written in a text editor) evidence, they still have the same level of security, but it is a behaviour that should be discouraged, especially as it is possible to perform the task mentally, as shown in the usability test.

The last attack is quite involved but extremely dangerous. Each time a user creates a password and sends it (unhashed) to a server, it gives the server some information. An adversary that would manage to obtain control of the server of a company that requires frequent changes would be able — after a delay — to obtain multiple passwords for all users. A few factors mitigate this, thankfully. First, password policies are on their way out [37]. Second, this requires either a hacker that is durably infiltrating a server or a malicious organisation that imposes password changes on its users (which they might not tolerate as such changes are mostly seen in company servers or banking systems [38]). Finally, all usable password composition practices today — short of randomly generated passwords — suffer from this issue, which is compounded by the fact that hashing is often done imperfectly [39].

D. Remaining threats

One smart way of attacking this system relies on finding a big intersection between two passwords. This mainly happens when two cues are extremely similar — e.g. sharing the first three letters — leading to very close passwords that can leave the adversary with only three characters left to guess. This could push some hackers to target the user data of services whose natural cues might be close to the ones of valuable services.

There is, however, a simple way to lower the risk: telling users to create their cue in a memorable way, while trying to avoid very similar cues: if they need cues for "GoDaddy" and "GoAir", they should choose `gdad` and `gair` for their cues, instead of `goay` and `goai`. This is reasonable, and a quick simulation shows that, even without changing a single letter, the median number of cues generated before a real collision is 79, quite above the number of accounts most users have at any point [9].

The scheme is *analysable*, as it is deterministic. We have also shown that the scheme is *publishable* and *secure* against most known attacks. We must now look at its usability.

V. USABILITY

Regardless of how secure a password is, if it is too hard to make, it will be reused. If it is too hard to remember, it might be stored in a possibly insecure way, such as on paper or in a file. Usability constraints of retrievability, low effort, and adaptability are all critical to the success of a scheme.

A. Retrievability

One of the biggest sources of online frustration [9] is forgetting a password and trying several plausible ones before abandoning and resetting it — when possible. This can be compounded by the fact that the next few proposed passwords might get discarded as they correspond to past passwords, pushing the user to create ever harder passwords, getting confused about which ones worked and so on, and forgetting them even more frequently. Moreover, frequent resets can pose security risks by themselves.

1) Passphrase: In the case of Cue-Pin-Select, the information needed to retrieve the passwords is easily memorable or retrievable, and generally both. The most important is the passphrase itself, which should be easy to remember by being quite short [40] and meaningful to the user. Three factors play a role in its memorability. First, allowing users to *create* their own passphrase from six given words by manipulating the order and having the opportunity to add their own words makes it more personal, safer in its extra length and still easier to remember, as in [29].

Second, more than 35% of users need a password for a new service at least once per week, and more than 90% need at least a few per year [9]. Repeated use of the scheme will serve as rehearsals and cement the passphrase in their memory.

Third, if the user never uses their passphrase or the generated passwords, it has low utility for them, and no scheme would truly work in such a case. However, there is the possibility of a user never creating new passwords after an initial period and memorising the ones they have. In such a case, those passwords could serve as strong information that would help their memory.

The passphrase becomes more memorable after the user has started using it. It is also retrievable if they forget part of it after a period of disuse. Together, the last two correspond to the *self-rehearsing* constraint mentioned in [5].

a) PIN: As can be seen from the global use of 4-digit numbers for credit cards, phone passwords, and door codes, people are used to and capable of remembering this kind of PIN. Despite this, some users could forget their PIN. In such a case, it would be easy to find it back using only the passphrase and one of their passwords. Finally, if some users struggle with numbers and might risk forgetting both the PIN and their passwords, we provide a variant that does not require it (albeit at a small entropy cost).

b) Cue: As the cue is short and users are discouraged from having a complex cue, it should be memorable. More importantly, it is retrievable as there are only a few imaginable cues each user could create from a given service, so a few tries would be enough to get the cue back. Also, it is possible to write down the list of cues in a file, as the security analyses don't assume that they are secret. This is especially true for the variant where the user has to change their password frequently.

c) Password: The password is the least memorable piece of information, being composed of 12 pseudo-random alphabetic characters. However, any secure password of such length will also be hard to remember, unless it shares strong similarities with other passwords (thus making it vulnerable). Despite this, thanks to the fact that it is created from parts of words and through associative memory and repetition, users should be able to remember their most frequently used passwords. Finally, the password is easily and entirely retrievable from the other pieces of information in a quick and deterministic fashion.

When a user forgets their password, which will happen, their first step is to rerun Cue-Pin-Select to obtain their original password. However, that might not work if they are starting from a wrong cue or have forgotten about special constraints. In such a case, the user just needs to look at the password constraints on the service they are using to figure whether they had added any special characters (which is a deterministic process). This means that if they know the constraints and their passphrase, the only unknown left is the cue, so at most a few tries would be needed.

B. Low effort

Initialising the password scheme, creating or retrieving a password, and entering it should all be tasks that are not difficult for users, in both time and effort.. If they are hard, the user will resent it or make mistakes as they have other goals for using services than simply securing them. There are also many cases where it is strongly advantageous to have no dependence on physical devices (such as when one is in public, forgets their computer, or tries to give their password on the phone). Hence, having a computer, or even a pen and paper, should only marginally help the user and it should be feasible to use the password scheme without those (see user study below). This corresponds to the constraints of *human usability* and *agent independence*.

The efforts needed for mental password managers can be split into three categories, as long as we're following the principle of *self-rehearsal*:

- Time and difficulty to learn how to use the scheme and initialise it (such as by creating a passphrase).
- Difficulty to remember and enter passwords once it is used in everyday life.
- Effort to get your password back when you lose it.

Generating the initial passphrase and PIN is easy, as one only needs to get six random words and a random PIN from any generator (some being findable online), and organise them in the order of their choice.

To create a new password, one starts by generating a cue which is immediate as it should be the first 4-letter string that comes into the user's mind. The rest consists of counting and moving 4 groups of 3 characters. It has only 3 steps between adding to the password being typed — 12 steps total.

It requires no mental arithmetics, so it should be accessible to people with dyscalculia and should not even require a piece of paper once the user is familiar with the system.

Passwords generated by Cue-Pin-Select should be easy to enter, being composed entirely of alphabetical characters (and when absolutely necessary the required one or two special characters), but the user must first remember the password, and this is not a given. We can distinguish three main cases:

- Frequently used passwords (entered at least every few days): they tend to be remembered by the users even in case of relatively high complexity. As this also applies to passwords created using Cue-Pin-Select, frequently used passwords should not suffer from increased effort or loss of performance.
- Rarely used passwords (entered at most a few times per year): they tend to be forgotten by the users, leading to frustration, many tries, and password resets. For Cue-Pin-Select, this only leads to password regeneration, which requires recomputing it from a small remembered — or stored — cue and is still lower effort compared to trying a few times before resetting the account.
- Passwords used with medium or variable frequency: users might forget them, and a simpler password could be more memorable than one generated with Cue-Pin-Select. However, Cue-Pin-Select can be used to regenerate the password without changing it, giving the user one more rehearsal opportunity. On the other hand, with the usual password schemes, having to reset it and pick a completely different one wipes out the previous effort made to remember it, even if it happens less frequently, making it costlier in the long term.

C. Adaptability

It then seems that for most users and use-cases, systematic use of Cue-Pin-Select should be easier. That, however, requires the scheme to be always applicable, no matter the constraints imposed by the service provider. It must then have no dependence on the password character set, length, and reuse change policy that a service imposes. Password requirement can also be contradictory between different services (like short maximal length constraints or forbidden numbers). Any exception that prevents the user from using one scheme drastically diminishes the interest in using that scheme. Among protocols that are now known to create usability errors [19], some still ask users to regularly change their passwords, avoiding any that have some similarity to ones used previously in a large time frame. Some users also forget their password despite their best efforts or make a typo while defining it for the first time. Adaptable password scheme must then include the possibility of creating new passwords for a single service without introducing confusion as to which one is the current password, as users dislike changing habits and will keep one scheme (or one password) for multiple years at a time.

The passwords created by Cue-Pin-Select heretofore have been in lower-case alphabetical characters. This provides enough security by itself but could be changed as needed to work with idiosyncratic password requirements. Even the most trivial extension that takes care of this for each of the following requirements does not reduce entropy:

- If the password requires capitalisation, the user should remember to capitalise one letter in their cue, and capitalise the corresponding three letters of their password.
- If it requires a number, the user can start with 0 and insert it at the center of their password, then increase it by one each time they renew this password.
- If it requires a special character, they can pick one in particular, like "!" that they will put at the end (or in the center) of every password that requires it.
- If it has a maximal length, they can just truncate the password without losing too much entropy (and a service that limits passwords to lengths smaller than 12 probably has bad security in any case).
- If it requires the user to change their password at set intervals of time (such as every month), without repetition for a certain time, they can change the first letter of the cue (or the first two letters) by cycling slowly over the alphabet. AMZN would become BMZN and then CMZN and so forth, and the passwords would be strongly different each time (with high probability), as it changes the starting point.

These simple changes give ways to adhere to the security constraints required by service providers without reducing the entropy of the password or significantly reducing usability.

VI. Testing Cue-Pin-Select

With strong arguments in favour of the usability of Cue-Pin-Select, a usability test was organised, with 11 subjects using it for short tasks each day for four days[4]. Their personal data was neither stored nor shared. We told them not to use the passwords generated, but encouraged them to take the benefit of learning this simplifying system for later use with their own passphrase and passwords. The group consisted of five men and six women of diverse non-scientific backgrounds, varying in ages from 18 to 65.

A. Protocol

Users were initially given a document explaining what they needed to know, including how to use Cue-Pin-Select. The document explained that they could leave at any time, that they should not use the passwords they would generate over the experiment as we would ask for that information, but that they were free to use the system with another passphrase after the experiment. They were progressively given three sets of tasks,

[4]Due to funding constraints, volunteers were used, which limited the duration of the experiment if we wanted a variety of users while limiting the amount of drop-outs. As we were writing this paper, we received anecdotal evidence from a colleague who was not involved in the development of the algorithm but started using it after we discussed it in the lab. They reported that after a few months, caching effects created through habits — such as going immediately to the corresponding letter in the passphrase from the cue and PIN — made the scheme much more efficient. Only account creation was still slow as they took care to avoid creating a wrong password or double-checked their computations. Even then, they self-reported that it seldom took more than 30s.

each lasting a few minutes. They were then told to follow the self-administered tasks at the rate of one in the morning and one in the afternoon, and send us the results, as well as the time it took them to accomplish each task. The list of tasks is as follows:

- Day 1, task 1: Create their passphrase and PIN (task 0). Create two passwords with cues already provided. Create cues and then passwords for two services (*Hotmail* and *Yahoo*). After this task, they were given feedback to explain potential errors in making a password they might have done.
- Day 1, task 2: Create a password with a provided cue, and then a (cue/password) couple for *New York Times*. Told to try to remember their passphrase, as they would have to recall it from memory from the second day onward.
- Day 2, task 1: Recall the passphrase, then create a password with a cue provided and a (cue/password) couple for *Twitter*.
- Day 2, task 2: Create a (cue/password) couple for *Snapchat*, and recall the one they did for *New York Times*.
- Day 3, task 1: Create 2 (cue/password) couples for *Reddit* and *AT&T*.
- Day 3, task 2: From this step on, the participants were told to apply the algorithm entirely in their head (writing down only the letters of their passwords as they computed them). Create 2 (cue/password) couples for *The Guardian* and *HP*.
- Day 4, task 1: Create 2 (cue/password) couples for *Spotify* and *Gmail*.
- Day 4, task 2: Recall the couples they created for *Snapchat*, *AT&T* and *HP*.

After the experiment, users were presented with questions asking them if they had trouble remembering their passphrases (instead of asking them if they cheated), which tasks they had done with pen and paper and which in their head (as it appeared that some switched earlier than in the instructions), whether certain aspects made them lose some time, and whether they would consider using it in its current state.

B. Results

Only one participant had trouble remembering their passphrase on the second day (they were missing one word) and had to be given it back. Most of them developed mnemonic schemes to help them remember (or increase their speed), such as splitting it into two sentences or creating mental imagery. A few had trouble remembering their cues.

Some users had trouble following the initial instructions (or found them unclear) as they were not very well explained. The most common mistake was restarting from the start of the word at each new cue. Feedback was given after the first set of tasks and first password to teach them to use Cue-Pin-Select correctly. By the second set of tasks, all users could correctly execute the algorithm (with only three recorded mistakes in that task and the following).

As is shown in Figure 4 — with detailed data in Figure 5 — there was a general speed-up over the course of the experiment. We can see a speed-up within each set of tasks, and a slowdown between sets, with a stronger slowdown on the afternoon of the third day, as all users had to compute passwords mentally and couldn't use pen or paper or their electronic device anymore[5]. As we can see from the end of the table, speeds were still increasing at the end of the fourth day, but that's when we had to stop the experiment. The real time taken by users who regularly use this method can then be inferred to be lower than the one attained at the end of the experiment, although we do not know by what margin.

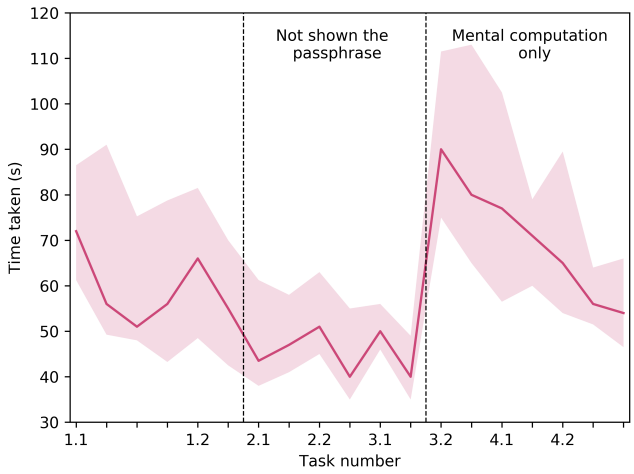

Fig. 4. Time taken by participants to create a password over the course of the experiments. The solid curve corresponds to the median time and the shaded area represents the time taken by people between the 25th and 75th percentiles. The sudden increase at 3.2 corresponds to the switch to mental-only tasks.

Task	1.1a	1.1b	1.1c	1.1d	1.2a	1.2b	2.1a	2.1b	2.2a	2.2b
Mean	89	82	72	63	70	59	50	49	54	45
Median	72	56	51	56	66	55	44	47	51	40
Max	233	211	222	108	132	113	87	68	70	61
Min	47	35	35	32	32	33	30	32	42	31

Task	3.1a	3.1b	3.2a	3.2b	4.1a	4.1b	4.2a	4.2b	4.2c
Average	51	42	105	86	81	74	67	58	57
Median	50	40	90	80	77	71	65	56	54
Max	74	53	220	131	130	117	106	86	71
Min	38	30	65	47	46	47	24	33	31

Fig. 5. Time taken by participants to create each password, in seconds. The top table has the times from the first and second days, and the bottom table has the third and fourth days. As in Figure 4, the increase at 3.2 is caused by the added rule that forbade participants from using pen and paper, having to compute the passwords mentally instead.

Users saw the algorithm's value, despite the lower case only, no special characters demonstration. Four out of eight users

[5]Two users decided to do all tasks mentally from day 2 onward, and their data was not counted in the tables for days 2 and 3, but they show the same learning behaviour as the others. Instead of writing down the passwords as they created it, some users tried to create all of it before writing it down; this data is included in the tables.

who gave feedback said they would use this system, at least for their important passwords. Two were hesitant; one thought that Cue-Pin-Select wasn't adaptable enough (indeed, they were not shown how to use capitalisation or special characters). Finally, one said they wouldn't use it as it didn't fit their personal security needs.

C. Feedback

Multiple participants observed that there was a strong cost in time (and usability) when they had a letter that was absent from their passphrase and had to go through their passphrase multiple times. They said that if they ever used the scheme again, they would make sure to have more vowels in their passphrases, as well as a higher letter diversity (which is also good from a security standpoint). Two of them also mentioned they would prefer a PIN with lower numbers.

VII. Conclusion

This paper has introduced the Cue-Pin-Select usable password creation scheme and analysed its security and usability. It can be learned and applied by a novice in less than two minutes and after making a few passwords, all testers were able to create 12-character retrievable passwords in under a minute in their head. Some participants were able to create or retrieve passwords in less than half a minute. The scheme is robust against many types of attacks, including against an adversary in possession of some of the generated passwords. All of the passwords in the scheme are easily retrievable assuming the participant remembers their passphrase. The passphrase itself is easily memorable, frequently rehearsed, and retrievable via associative memory if the participant remembers some of the passwords. It allows a participant to create a cue that works for them. Finally, it is compatible with text-based password constraints and can be used durably without frustration or risk from an evolving constraint environment. In summary, it satisfies [5]'s five criteria of *analysability*, *publishability*, *security*, *self-rehearsal* and *human usability*, as well as our criteria of *agent independence*, *adaptability*, and *scalability*. When compared with the previous work, those three properties are crucial as they mean that improvements do not necessitate a complete overhaul of web service authentication systems.

The analysis performed gives worst-case lower bounds on the security of Cue-Pin-Select, by making strong assumptions on the amount of information available to the adversary. Instead of having clear-text passwords, analyses supposed that they knew the length of the passphrase and the location of each letter, while a significant chunk of the security of the scheme relies on their secrecy. Despite those assumptions, the system guarantees 40-bit security even against single plain-text attacks. Those bounds could be increased by a more thorough analysis of realistic attacks, to prove a higher level of security against multiple plain-texts attacks.

This exercise had several goals. It shows the existence of an easy-to-use password generation and retrieval system. It gains security from not requiring a computer, from the entropy of a memorable secret, and from its adaptability. It gains usability by being personalised, based on language, and by rehearsing the only things that have to be remembered. It gains robustness in the ease it gives for retrieving any passwords a user does not remember, and for giving the users simple rules to make up alternative passwords for any service. With a greater speed and no reliance on mathematical primitives, the system is easier to use and more secure than earlier proposals [5].

That said, some questions remain open. Getting accurate values on the usability would be welcome, by having experiments with both more users, and longer total duration. Seeing how those users would handle the variant that addresses idiosyncratic constraints and how much of a cost it creates would also be interesting. Sadly, it is hard to compare it directly to previous proposals, as they do not tend to address the adaptability constraint. However, different schemes could be developed to allow for comparative studies.

In a different direction, the strong security constraints could be relaxed to get a more usable scheme with a slightly weaker but still strong security features. A thorough analysis of more realistic threat models against Cue-Pin-Select and derived schemes that rely less on private information held by the adversary would help in this endeavour. The Appendix after the references shows multiple variants that could make such analyses easier.

References

[1] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse." in *NDSS*, vol. 14, 2014, pp. 23–26.

[2] P. Lipa, "The security risks of using "forgot my password" to manage passwords," 2016, accessed: 2017-12-18. [Online]. Available: https://web.archive.org/web/20170802185615/https://www.stickypassword.com/blog/the-security-risks-of-using-forgot-my-password-to-manage-passwords/

[3] N. J. Hopper and M. Blum, "Secure human identification protocols," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2001, pp. 52–66.

[4] J. Blocki, M. Blum, and A. Datta, "Naturally rehearsing passwords," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2013, pp. 361–380.

[5] M. Blum and S. S. Vempala, "Publishable humanly usable secure password creation schemas." in *3rd AAAI Conference on Human Computation and Crowdsourcing*, 2015.

[6] H.-M. Sun, Y.-H. Chen, and Y.-H. Lin, "oPass: A user authentication protocol resistant to password stealing and password reuse attacks," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 2, pp. 651–663, 2012.

[7] J. Blocki, M. Blum, A. Datta, and S. Vempala, "Towards human computable passwords," *8th Innovations in Theoretical Computer Science Conference – ITCS 2017*, 2017.

[8] U. Topkara, M. J. Atallah, and M. Topkara, "Passwords decay, words endure: Secure and re-usable multiple password mnemonics," in *Proceedings of the 2007 ACM Symposium on Applied Computing*, ser. SAC '07. New York, NY, USA: ACM, 2007, pp. 292–299.

[9] Centrify, "Centrify password survey: Summary," Centrify, Tech. Rep., 2014, accessed: 2017-12-20. [Online]. Available: https://www.centrify.com/resources/5778-centrify-password-survey-summary/

[10] Lastpass, "Psychology of passwords survey," Lastpass, Tech. Rep., 2016.

[11] M. S. A. N. Ranak, S. Azad, M. Safwan Fathi Bin, Z. Kamal, and M. Rahman, "An analysis on vulnerabilities of password retrying," *5th International Conference on Software Engineering & Computer System*, 2017.

[12] G. Stewart and D. Lacey, "Death by a thousand facts: Criticising the technocratic approach to information security awareness," *Information Management & Computer Security*, vol. 20, no. 1, pp. 29–38, 2012.

[13] J. Abawajy, "User preference of cyber security awareness delivery methods," *Behaviour & Information Technology*, vol. 33, no. 3, pp. 237–248, 2014.

[14] N. Alkaldi and K. Renaud, "Why do people adopt, or reject, smartphone password managers?" in *Proceedings of EuroUSEC*, 2016, eprint on Enlighten: Publications.

[15] D. Florêncio, C. Herley, and P. C. van Oorschot, "Password portfolios and the finite-effort user: Sustainably managing large numbers of accounts," in *23rd USENIX Security Symposium*. San Diego, CA: USENIX Association, 2014, pp. 575–590. [Online]. Available: https://web.archive.org/web/20170823094633/https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/florencio

[16] F. Mwagwabi, T. McGill, and M. Dixon, "Improving compliance with password guidelines: How user perceptions of passwords and security threats affect compliance with guidelines," in *47th Hawaii International Conference on System Sciences – HICSS*, vol. 00, 1 2014, pp. 3188–3197.

[17] D. E. Kieras and S. Bovair, "The role of a mental model in learning to operate a device," *Cognitive Science*, vol. 8, no. 3, pp. 255–273, 1984.

[18] R. Shay, S. Komanduri, A. L. Durity, P. S. Huh, M. L. Mazurek, S. M. Segreti, B. Ur, L. Bauer, N. Christin, and L. F. Cranor, "Can long passwords be secure and usable?" in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '14. New York, NY, USA: ACM, 2014, pp. 2927–2936.

[19] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman, "Of passwords and people: Measuring the effect of password-composition policies," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 2595–2604.

[20] S. Gaw and E. W. Felten, "Password management strategies for online accounts," in *Proceedings of the Second Symposium on Usable Privacy and Security*, ser. SOUPS '06. New York, NY, USA: ACM, 2006, pp. 44–55.

[21] Z. Li, W. He, D. Akhawe, and D. Song, "The emperor's new password manager: Security analysis of web-based password managers," in *23rd USENIX Security Symposium*. San Diego, CA: USENIX Association, 2014, pp. 465–479. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/li_zhiwei

[22] N. Memon, "How biometric authentication poses new challenges to our security and privacy [in the spotlight]," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 196–194, 2017.

[23] B. Choudhury, P. Then, B. Issac, V. Raman, and M. Haldar, "A survey on biometrics and cancelable biometrics systems," *International Journal of Image and Graphics*, vol. 18, p. 1850006, 01 2018.

[24] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 553–567.

[25] C. Kuo, S. Romanosky, and L. F. Cranor, "Human selection of mnemonic phrase-based passwords," in *Proceedings of the second symposium on Usable privacy and security*. ACM, 2006, pp. 67–78.

[26] K. Lee, "Four methods to create a secure password you'll actually remember," 2014, accessed: 2017-12-18. [Online]. Available: https://web.archive.org/web/20190123014846/https://www.lifehacker.com.au/2014/07/four-methods-to-create-a-secure-password-youll-actually-remember/

[27] M. Blum and S. Vempala, "The complexity of human computation: A concrete model with an application to passwords," *CoRR*, vol. abs/1707.01204, 2017. [Online]. Available: http://arxiv.org/abs/1707.01204

[28] S. Samadi, S. Vempala, and A. T. Kalai, "Usability of humanly computable passwords," in *6th AAAI Conference on Human Computation and Crowdsourcing*, 2018.

[29] N. K. Blanchard, C. Malaingre, and T. Selker, "Improving security and usability with guided word choice," *34th Annual Computer Security Applications Conference – ACSAC*, 2018.

[30] P. Norvig, "Natural language corpus data," *Beautiful Data*, pp. 219–242, 2009.

[31] L. Q. Ha, E. I. Sicilia-Garcia, J. Ming, and F. J. Smith, "Extension of zipf's law to words and phrases," in *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, ser. COLING '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 1–6.

[32] D. I. Eastlake, J. Schiller, and S. Crocker. (2005) Rfc4086: Randomness requirements for security. [Online]. Available: https://tools.ietf.org/html/rfc4086

[33] "Amazon AWS S3 cost calculator," Amazon, 2018, accessed: 2017-12-18. [Online]. Available: https://calculator.s3.amazonaws.com/index.html

[34] P. Grassi, M. Garcia, and J. Fenton, "NIST special publication 800-63-3," *Digital Identity Guidelines*, vol. 1, 2017.

[35] A. Toponce, "Password cards," 2010, accessed: 2017-12-18. [Online]. Available: https://web.archive.org/web/20121101053014/http://pthree.org/2010/09/21/password-cards/

[36] ——, "Strong Passwords NEED Entropy," 2011, accessed: 2017-12-18. [Online]. Available: https://web.archive.org/web/20180223215746/https://pthree.org/2011/03/07/strong-passwords-need-entropy/

[37] L. Spitzner, "Time for password expiration to die," 2019, accessed: 2019-10-03. [Online]. Available: https://web.archive.org/web/20190709132850/https://www.sans.org/security-awareness-training/blog/time-password-expiration-die

[38] H. Habib, P. E. Naeini, S. Devlin, M. Oates, C. Swoopes, L. Bauer, N. Christin, and L. F. Cranor, "User behaviors and attitudes under password expiration policies," in *Fourteenth Symposium on Usable Privacy and Security ({SOUPS} 2018)*, 2018, pp. 13–30.

[39] N. K. Blanchard, X. Coquand, and T. Selker, "Moving to client-sided hashing for online authentication," in *Workshop on Socio-Technical Aspects in Security and Trust – STAST*, 2019.

[40] G. A. Miller, "The magical number seven, plus or minus two: some limits on our capacity for processing information." *Psychological review*, vol. 63, no. 2, p. 81, 1956.

VIII. Appendix: variants of Cue-Pin-Selec

As it is easy to add vulnerabilities, each variant must be systematically analysed. The main variant of Cue-Pin-Select is shown in Algorithm 2. It incorporates the modifications mentioned above to make it compatible with most online constraints, not requiring any change in the security analysis.

A. Lower security variants

There are many other ways to create variants, and most are problematic, implying that Cue-Pin-Select might be a local optimum in the security-usability trade-off.

a) Fewer words or fewer characters: Reducing the passphrase from 6 to 5 might be tempting. Computer simulations using the same protocol as before show that with this approach, many single-plain-text attacks would leave the user with fewer than 28 bits of security, down to 16 bits in certain cases. Fewer characters in the password would be imaginable but would lower the individual resistance, and the number of possibilities would be limited as:

- Having 3 passes in the algorithm would lower the entropy too much.
- Extracting 1 or 2 characters would require more passes, and be less user friendly.

b) Getting rid of the PIN: Getting rid of the PIN might seem to simplify the algorithm. This is a dangerous idea; as described above, this PIN plays an important part in making the algorithm secure. As cues can be easily guessable by an adversary, they could accumulate way more information on the passphrase by guessing the cues (and then could perform targeted attacks to obtain the rest).

If the problem is not using the PIN but remembering it, a viable password generation scheme with a further reduction on memory would come from making the PIN from the passphrase. This alternative that strongly mitigates the risk is

to have a PIN that corresponds to the lengths of the last four words (modulo 10). This maintains a high level of security while making the PIN trivial to retrieve.

Algorithm 2: Adaptable Cue-Pin-Select

Data: Passphrase $PHRASE$ of at least 6 random words
PIN P of 4 random digits
service name $NAME$
/* If the user hates numbers, the PIN can be the length of the last 4 words */
Result: string S of about 12 characters
begin
  From $NAME$ create string CUE of four characters
  /* This user-chosen string should be easy to remember */
  if *Service requires mixed-case* **then**
    CUE should be in mixed-case
  if *User had a previous cue for this service* **then**
    $CUE[0]$ and $CUE[2]$ become the next letters in the alphabet
  $LEN \longleftarrow$ Length$(PHRASE)$
  $INDEX \longleftarrow 0$; $S \longleftarrow []$
  for $i = 0$; $i < 4$; $i++$ **do**
    $LETTER \longleftarrow CUE[i]$
    while $LETTER \notin PHRASE$ **do**
      $LETTER \longleftarrow$ letter following $LETTER$ in the alphabet
    $INDEX \longleftarrow$ next occurrence of $LETTER$ in $PHRASE$ after $INDEX$
    /* Or the first occurrence if we reach the end of $PHRASE$ */
    $INDEX \longleftarrow (INDEX + PIN[i] + 3) \mod LEN$
    $TEMP \longleftarrow$ Concatenate $(PHRASE[INDEX-2, INDEX-1, INDEX])$
    if $CUE[i]$ *is upper-case* **then**
      Make $TEMP$ upper-case
    $S \longleftarrow$ Concatenate $(S, TEMP)$
  if *service requires a number* **then**
    $S \longleftarrow$ Concatenate $(S, 0)$
  if *service requires a special character* **then**
    $S \longleftarrow$ Concatenate $(S, !)$
  if *service requires a password of length* $LEN - y$ **then**
    $S \longleftarrow$ Suffix $(S, LEN - y)$
  /* We avoid the security measures by adding characters that don't change entropy, and remove the first few letters if needed */
  Print S

c) Using parts of words: Instead of extracting trigrams from the passphrase, it would be simpler to extract larger character-groups from words one at a time. For example, one could take a random letter and the corresponding prefix or suffix to create those groups, which could be faster and easier for the user. Unfortunately, this would reduce entropy so much[6] that it would either require more words, more passes or make the passphrase itself insecure, depending on how many letters are extracted each time.

[6]The multiple potential methods differ somewhat, but the common loss in entropy is due to two factors. The first is that the set of prefixes and suffixes is in fact not that large when compared to trigrams that can include part of a suffix and part of a prefix. The second is the very variable length of prefixes ans suffixes, meaning that some would be composed of a single letter, drastically diminishing entropy, or instead be very long (up to 7 characters), diminishing usability while only marginally increasing entropy.

B. Higher security variants

A few variants have either a higher degree of security or more easily provable and analysable security, coming at the cost of a reduced usability.

Algorithm 3: Higher Security Cue-Pin-Select

Data: Passphrase $PHRASE$ of at least 8 random words
PIN P of 4 random digits
service name $NAME$
Result: string S of 12 characters
begin
  From $NAME$ create string CUE of four characters
  /* This user-chosen string should be easy to remember */
  $INDEX \longleftarrow 0$
  $L \longleftarrow$ Length(P)
  $S \longleftarrow []$
  for $i = 0$; $i < 4$; $i++$ **do**
    $LETTER \longleftarrow$ Integer$(CUE[i])$
    /* $LETTER \longleftarrow$ n, if $CUE[i]$ is the n-th letter in the alphabet */
    $INDEX \longleftarrow INDEX + LETTER + PIN[i] + 3 \mod (L)$
    $S \longleftarrow$ Concatenate $(S, PHRASE[INDEX - 2, INDEX - 1, INDEX])$
  Print S

a) Using letter-values: The first version of this algorithm behaved differently in the Cue step. Instead of looking for the next letter identical to the one in the cue, the user was supposed to convert the cue into a number, and then advance by that many characters (plus the PIN digit). This means that the distribution of trigrams is even closer to a uniform one, and also means that any modification to the first letter of the cue entirely changes the rest of the password. However, the mental load associated with converting a letter to a number and moving by more than 15 characters on average slows down the scheme and makes it more complex to explain and perform. It is still shown in Algorithm 3 for numerically adept users.

b) Using more words: Finally, we could imagine having the same scheme but extracting $k \geq 4$ trigrams from $m > 6$ words. This is obviously less usable, but it increases security by a huge factor, especially in the case of multiple plain-text attacks (when we increase m). For example, going to $m = 8$ guarantees high entropy against triple plain-text attacks. Experimentally, every added word increases entropy by 16 bits, and every added plain-text reduces entropy by less than 20 bits, although those two procedures also increase the entropy variance. This is only true as long as the algorithm still goes through the whole passphrase more than once on average, so adding words until the passphrase has more than 60 characters is a good compromise. Adding more words afterwards would generally be counterproductive, unless $k \geq 4$.