Cue-Pin-Select, a Secure and Usable Offline Password Scheme

Leave Authors Anonymous
for Submission
City, Country
e-mail address

Leave Authors Anonymous
for Submission
City, Country
e-mail address

Leave Authors Anonymous
for Submission
City, Country
e-mail address

ABSTRACT

People struggle to invent safe passwords for many of their typical online activities. This leads 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 [10, 18]. Recent research [14, 6] has offered multiple alternatives but those require either rote memorization [8] or computation on a physical device [23, 7].

This paper presents the Cue-Pin-Select password family scheme, which uses simple mental operations (counting and character selection) to create a password from a passphrase and the name of the service the password is targeted for. It needs little memorization to create and retrieve passwords, and requires no assistance from any physical device. It is durable and adaptable to different password requirements. It is secure against known threat models, including against adversaries with stolen passwords. A usability test shows the successes of users in real-life conditions over four days.

ACM Classification Keywords

D.4.6 Security and Protection: Authentication; H.5.2 User Interfaces: User-centered design

Author Keywords

Usable-security; Password schemes; Authentication.

INTRODUCTION

The usable password community has shown that people tend to choose passwords that are easy to rehearse and remember [11]. Strategies of requiring long passwords or multiple types of characters (mixed-case, numbers and symbols) have been shown by Cranor and others [22, 16] to often be misleading, not improving security in practice. These password policies seldom address the issue of entropy as most of them can be gamed with an easy to guess password like *1Monkey!!* [19, 25]. In addition, encouraging people to make passwords in the moment, and in an ad-hoc way, can make people prone to simply complying with the policies without creating systematically secure passwords. Those new passwords are not only weak, but also forgettable, and lead to frequent resets of rarely used accounts.

Moreover, the proliferation of online accounts per user [1] means that, due to memory constraints, passwords tend to be either handled by a password manager [11, 15] – or worse, written down – or reused [26]. In that last case, the complexity that users tolerate for their main password often means that they will use it for many different online accounts. This greatly increases the points of failure, as a breach in the servers of any of those can compromise millions of identities [3]. In the other case, password managers introduce a major risk in case the device is stolen, constituting a single point of failure from which an adversary can steal a complete identity [2].

Password generation schemes provide a potential solution to those problems, depending on their design. Those schemes, which create new passwords automatically, can be arbitrarily simple or complex, and many schemes of varying strength have been devised. Those schemes often rely on our associative memory, which might be subject to interference, priming, nervous lapses, and attrition. As we are required to make many passwords for many services, we may sometimes produce families of related passwords which an adversary can feasibly infer if they learn some examples (i.e. Mypassword1! Mypassword2...). Potential solutions exist both in software and in published strategies for selecting good passwords, as suggested by Lifehacker [17] and others [8]. Such proposed simple improvements for generating or remembering passwords might make them more secure, and possibly easier for humans to remember as well.

In a series of papers, Blocki, Blum, Datta, and Vempala [6, 7, 8] have framed and formalized some important issues of Human Usability Models for passwords. The question of security can be measured by entropy (the numerical uncertainty of finding a specific code), access to the codes, and learnability of the families based on knowing one or more of the passwords. To show it can be done, they provide several approaches for remembering something as part of a method for making human computable families of passwords. For example, their first proposal, "DS1", requires a user to memorize a random letter-to-digit map as the seed to their password creation. Their "DS2" and "DS3" algorithms are similar with a slightly different mapping approach. They then describe a more conceptual

CHI'16, May 07–12, 2016, San Jose, CA, USA

ISBN 123-4567-24-567/08/06...$15.00

DOI: http://dx.doi.org/10.475/123_4

"WS1" that asks users to memorize both letters and words about a topic. Finally, they describe "LP1" that requires users to memorize a random letter permutation. These approaches are complex and will not pass the test of being quick or easy to use by people without paper.

In [8], the authors propose five criteria that all password schemes should respect:

- **Analyzable**: that the scheme follow a deterministic algorithm whose security can be assessed.
- **Publishable**: that the scheme is completely public (a version of Kerckhoff's principle[1].).
- **Secure**: that even an adversary with higher means cannot crack a user's password.
- **Self-rehearsing**: that the only training needed to remember all of one's passwords should come from normal use of the scheme itself.
- **Humanly usable**: that a human can easily learn and apply the scheme within reasonable time bounds.

The methods introduced in [8] and the one in [7] satisfy the first four, but not the usability requirement, as they take several minutes to create (or recreate) a password. Moreover, the ones from [6, 7] require machine computation, leaving the user dependent on a tool that could be complex to use and might not always be on hand.

Motivated by the need to make so many passwords and to retrieve them easily, this paper introduces a system, Cue-Pin-Select, that also fulfills additional usability and security criteria in addition to those listed above:

- **Agent-independent**: the scheme should be doable in one's head without needing assistance from a physical or computational system.
- **Durable**: the scheme should be adaptable to idiosyncratic requirements, including the ones that require users to frequently change their passwords. It should also be compatible with existing technologies.
- **Scalable**: the scheme should stay secure even for a large number of passwords.

The rest of the paper follows the following structure. The Cue-Pin-Select protocol is described and analyzed for its resistance to various threat models. A multi-use experiment is used to test usability over several days. Finally, multiple variants are considered to explain the design choices leading to Cue-Pin-Select.

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.

[1]A cryptographic scheme should depend on a key, and not on a secret algorithm

Passphrase

The main secret data has two components. The first one is a passphrase of at least 6 English words, and the second is a 4-digit PIN of the kind that people are accustomed to associating with bank cards. To generate the passphrase, the user is supplied with a randomly-generated sequence of 6 words, adding words (such as articles or connectors) if they so desire. The 4-digit number is also randomly generated.

A simple online tool (anonymized link) was also created that can be used to help create a high-entropy memorable passphrase. In this tool and the analyses, a special dictionary crafted for the purpose is used. This one started with a the first third of Peter Norvig's 300000 most frequent ngrams [21]. Those 100000 words still included words from other languages such as "unglaublichen" as well as some non-words like "unix-compile", which were removed. The intersection with the SOWPODS (list of admissible words in English Scrabble tournament) was then computed to obtain a list of the 87691 most frequent correct English words. This high number is further explained in the usability section.

Password generation algorithm

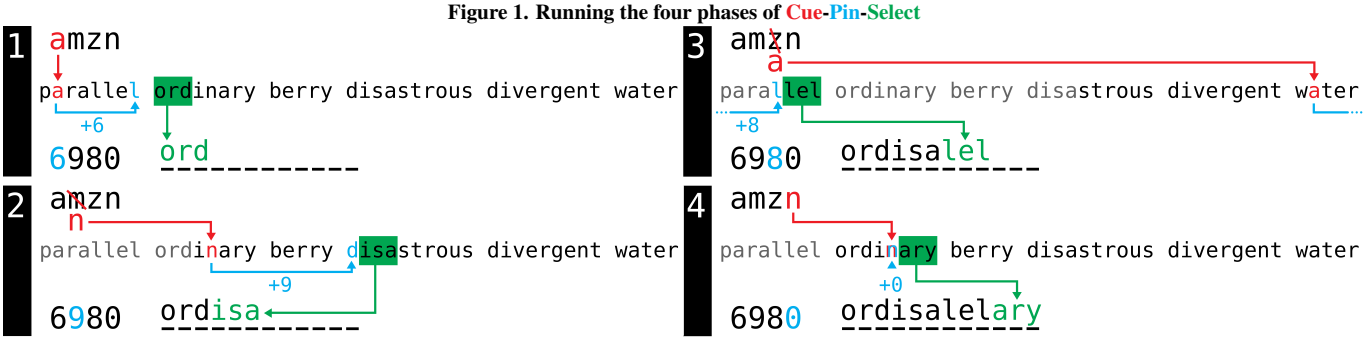
Each time the user needs a password for a new service, they need only apply the Cue-Pin-Select algorithm.

Data: Passphrase P of at least 6 random words
PIN K of 4 random digits
service name N
Result: String S of 12 characters
1 **begin**
2 From N, create string M of four characters
/* User-chosen, which should be easy to remember */
3 $L \longleftarrow$ Length(P)
4 $V \longleftarrow 0$
5 $S \longleftarrow$ ""
6 **for** $i = 0$; $i < 4$; $i++$ **do**
7 $X \longleftarrow M[i]$
8 **while** $X \notin P$ **do**
9 $X \longleftarrow$ letter following X in the alphabet
10 $V \longleftarrow$ index of next occurrence of $X \in P$ after V
/* Or wrap around to the first occurrence if the end of P is reached */
11 $V \longleftarrow V + K[i] + 3 \mod (L)$
12 $S \longleftrightarrow$ Concatenate $(S, P[V-2], P[V-1], P[V])$
/* All the indices are modulo L */
13 Print S

Algorithm 1: Cue-Pin-Select

The algorithm makes a 12-character password in 12 steps. Say a user has created the passphrase *parallel ordinary berry disastrous divergent water* and that their PIN is *6980*. 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, like *amzn*. This cue will then be used to extract password parts from the passphrase.

Figure 1. Running the four phases of Cue-Pin-Select



Once this is done, they look for the first letter of their cue in the passphrase. 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 *parallell*. They add the next three letters to their password, 'ord'. If the letter in the cue isn't in the passphrase, they look for the next letter in the alphabet 'b' for this first case (or the next after that, 'c' if there is no 'a' or 'b' in their passphrase).

They repeat the above four indexing steps three more times, for the second, third, and fourth number in their pin, starting each time from the last letter they used in the passphrase and wrapping to the begining of the passphrase when they index off the end of it. Figure 1 shows the process where each operation corresponds to a color, creating the password *ordisalelary*.

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 few tries, from which they can derive the whole password.

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.

SECURITY ANALYSIS

As 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, analysis here relies 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.

Preliminary considerations

One of the main assumptions used in the following analyses is that the distribution of three-letter chunks composing each password is very close to the distribution of a random chunk taken in a random passphrase. The PIN is an essential part of the randomization 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 chunks 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 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 3 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 nth 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, centered 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 preceeding or following the third trigram are not too far apart, and the same can be said for all the other pairs. Figure 2 shows the logscale distribution of passphrase lengths (exact computation based on our dictionary).

Scalability

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 specialized attacks targeting it. This is not true for a scheme used by millions of people. As such, all the threat models should assume Kerckhoff's principle that only the key (cue, PIN, and service code) are private, the algorithm being public. This corresponds to [8]'s notion of *publishable*. In our situation, there could actually be a 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.

The second type of scalability corresponds to the number of passwords used by a single user; frequently using a scheme

[2]The shape of those curves might seem to follow Zipf's law [12], 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.

Figure 2. Distribution of passphrase length on k words in vertical logscale

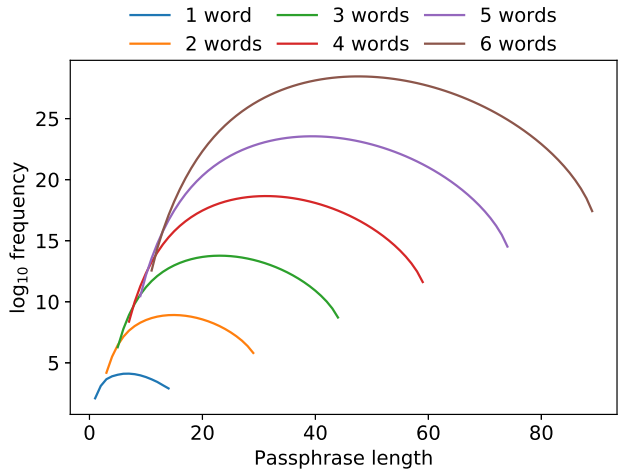
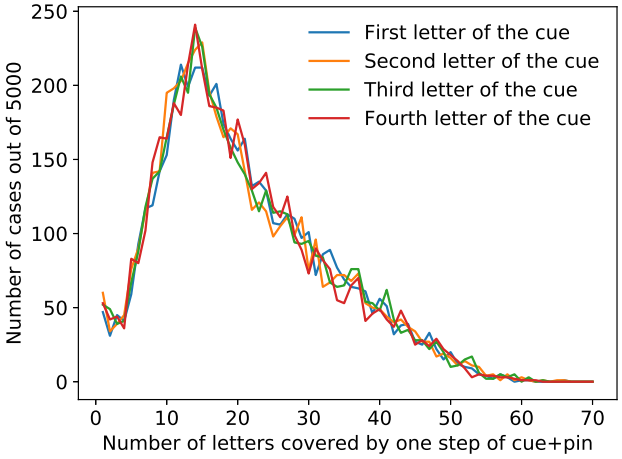


Figure 3. Distribution of length covered in one step



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 unscaleable 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. It's clear that it should be the case 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), assuming they already possess one of the two closest passwords.

Brute-force and dictionary attacks

Attacking the password

Current entropy recommendations against brute-force attacks vary from 29 bits to 128 bits of security, depending on the attack model [5]. One common recommendation proposes 36 bits of security on any given password for web services; such a password would require 1000 tries per second for one year to break. Assuming the attacker uses online servers to distribute the attack, in 2018 this would require more than $1000 per password, even with strong economies of scale [4].

In our case, assuming the adversary knows the scheme used, a strong 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. This is highest for prefix trigrams with 14 bits, close to 13 bits for central subwords and lowest for suffix trigrams with 11.4 bits. Thus, 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 bits of entropy.

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 87691, 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. This value is reduced by two factors: 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. 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.

Resistance to plain-text attacks

Plain-text attacks are one of the main vulnerabilities found in most user behaviors today, generally stemming from password reuse. This is also where typical methods as described by LifeHacker fail [17]. 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 [11].

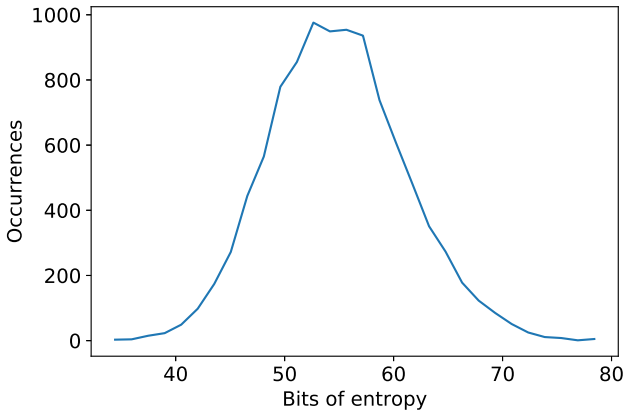
The scheme was designed to provide high security even in the event that one (or even a few) of the passwords are compromised, which can happen independently of the user's best practices. As said earlier, trying to guess one password from another in Cue-Pin-Select is a hard problem in the general case,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

Plain-text attacks

To analyze 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) on the following histogram:

Figure 4. Entropy left with a single plain-text

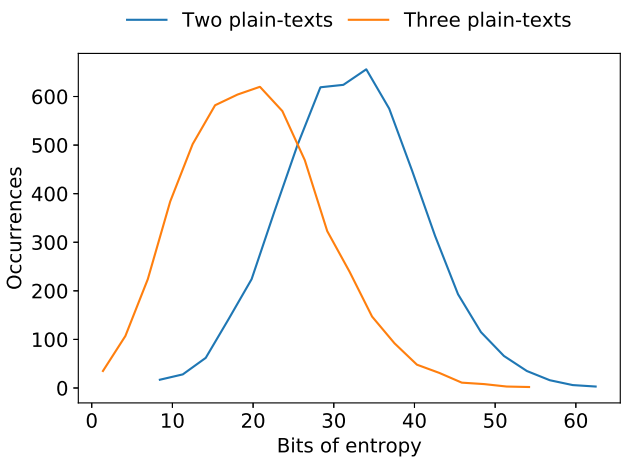


This shows that Cue-Pin-Select can guarantee a minimum of 40 bits of entropy in case of a plain-text attack, with an average of 54 bits (the standard deviation is 6 bits), and a maximum of 79 bits.

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 Figure 5 (5×10^3 runs each).

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 5000 simulations of the adversary

Figure 5. Entropy left with two and three plain-texts



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.

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 [6, 7], requiring semi-secure computation in the form of challenges from a computer. It can also include hybrid methods such as the password card [24, 25], in which obtaining the physical card does not give complete access but reduces the entropy of all 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 two 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 behavior that should be discouraged, especially as it is possible to perform the task mentally, as shown in the usability test.

Remaining threats

One main way of attacking this system relies on finding a big intersection between two passwords. This mainly happens when two cues are extremely similar, 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, as users can create more than 600 cues before they run out of high-security possibilities.

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

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 durability are all critical to the success of a scheme.

Retrievability

One of the biggest sources of online frustration[1] 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.

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 [20] 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.

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 [1]. 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 memorizing 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 [8].

There is one risk, however: if the user does not know the words presented during the passphrase generation, they will be hard to remember. There is a chance that some words in the 87691 word dictionary would be extremely unfamiliar to the user. Two factors mitigate this:

- The dictionary includes many words derived from others, forming what are called word families. For example, family, families, familiar, familiarity, unfamiliar, and 23 others share the same root. Estimates on the number of words known vary from 27 000 to 52 000 for native speakers [9], going down to less than 10 000 for non-native speakers. However, those studies look at *lemmas* and do not include many derived forms (for those, the studies show a vocabulary as high as 215 000 words for native English speaking undergraduate students [13]).

  Most speakers, then, should be familiar with the words of the dictionary despite its size.

- If the user cannot easily create a sentence or use some other way they are comfortable with to remember the words, they have the opportunity of restarting the passphrase generation without having any real impact on the entropy of passwords they will eventually create using it.

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).

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.

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.

Low effort

Initializing the password scheme, creating or retrieving a password, and entering it should all be tasks that are not difficult in time or effort for users. 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*.

Below we analyze the different kinds of efforts for the approach:

- Time and difficulty to learn how to use Cue-Pin-Select and create the passphrase and first passwords.
- Difficulty to remember and enter passwords once Cue-Pin-Select is used in everyday life.
- Effort to get your password back when you lose it.

Generating the initial passphrase is easy, as one only needs to get six random words and a random PIN from a any generator (some being findable online), and organize 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 by themselves should be easy to enter, being composed entirely of alphabetical characters (and sometimes 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) tend to be remembered by the users even in case of relatively high complexity, which also applies to passwords created using Cue-Pin-Select, leading to no increased effort or loss of performance.
- Rarely used passwords (entered at most a few times per year) 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 cue and is still low-effort.
- Passwords used with medium or variable frequency might be forgotten by users, and a simpler password would have a higher resistance to this risk. 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.

Durability

To be usable, schemes must be adaptable and applicable regardless of 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 [16], some still ask users to regularly change their passwords, avoiding any that have some similarity to ones used previously in a large time frame. Also, it happens that the user forgets their password despite their best efforts, which can even arise when they make a typo while defining it to the service. A durable 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 most 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 in this paper 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 capitalization, the user should remember to capitalize one letter in their cue, and capitalize 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 (such as a year or two), 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.

TESTING CUE-PIN-SELECT

With strong arguments in favor of the usability of Cue-Pin-Select, a usability test was organized, with 11 subjects using it for short tasks each day for four days. Their personal data was neither stored nor shared. We encouraged them to take the benefit of learning this simplifying system for later use for their own passwords. The group consisted of five men and six women of diverse backgrounds, varying in ages from 18 to 65.

Protocol

Participants 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, 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. 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.

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[3] (or found them unclear). 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 an occasional mistake).

All users sped up with a strong learning effect shown in the following figures 6, 7 and 8. Tasks on the afternoon of the third day took longer, as all users switched and couldn't use pen/paper or their electronic device anymore[4].

Figure 6. Time taken day 1

Time(s)	Task 1.1a	1.1b	1.1c	1.1d	1.2a	1.2b
Average	89	82	72	63	70	59
Median	72	56	51	56	66	55
Max	233	211	222	108	132	113
Min	47	35	35	32	32	33

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

[3] instructions have since been simplified

[4] 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 behavior 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.

Figure 7. Time taken day 2 and morning 3

Time(s)	Task 2.1a	2.1b	2.2a	2.2b	3.1a	3.1b
Average	50	49	54	45	51	42
Median	44	47	51	40	50	40
Max	87	68	70	61	74	53
Min	30	32	42	31	38	30

Figure 8. Time taken afternoon 3 and day 4 (mental tasks only)

Time(s)	Task 3.2a	3.2b	4.1a	4.1b	4.2a	4.2b	4.2c
Average	105	86	81	74	67	58	57
Median	90	80	77	71	65	56	54
Max	220	131	130	117	106	86	71
Min	65	47	46	47	24	33	31

Finally, one said it didn't fit their personal security/usability needs and they wouldn't use it.

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.

VARIANTS OF THE ALGORITHM

A first extension of Cue-Pin-Select as introduced above is shown in the Appendix, along with another variant with improved security.

As it is easy to add vulnerabilities, each variant must be systematically analyzed. Here are some natural variants with serious flaws.

Lower security variants

Fewer words or fewer characters

Reducing the passphrase from 6 to 5 might be tempting. Direct analysis shows 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.

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.

Using chunks of words

Instead of extracting chunks from the passphrase, it would be simpler to extract chunks from words one at a time. For example, one could take a random letter and the corresponding prefix or suffix to create those chunks, which would be faster and easier for the user. Unfortunately, this would reduce entropy so much that it would either require more words, more passes, or make the passphrase itself unsecure, depending on how many letters are extracted each time.

Higher security variants

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

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 chunks 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.

Using more words

Finally, we could imagine having the same scheme but extracting $k \geq 4$ chunks 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.

DISCUSSION

This paper has proposed, analyzed, and tested the Cue-Pin-Select usable password creation scheme. 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 users 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 user remembers

their passphrase. The passphrase itself is easily memorable, frequently rehearsed, and retrievable via associative memory if the user remembers some of the passwords. It allows a user 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. In summary, it satisfies [8]'s five criteria of *analyzability*, *publishability*, *security*, *self-rehearsal* and *human usability*, as well as our criteria of *agent independence*, *durability*, and *scalability*.

Users found it took a few minutes to learn at first but quickly gained proficiency. They had next to no trouble remembering their passphrase thanks to the multiple rehearsals. Working with the system for a few minutes each day, they all gained speed and accuracy in using the password creation algorithm. By the third day, all of them could create a password in under 55 seconds, and most could do it in their head in less than a minute by the end. This does not mean that they would spend 55 seconds each time they need to enter it, as any frequently used password would be itself remembered, requiring no recomputation. The strength of the scheme lies in having secure, diverse, and memorable passwords, that can be quickly recomputed even after a long time.

This exercise had several goals. It shows the existence of an easy-to-use password generation and retrieval system. It gains security from not using a computer, from its entropy, from the diversity of parts of its seed, and from its adaptability. It gains usability by being personalized and 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 user simple rules to make up alternative passwords for any service.

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. 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 paper shows that the new criteria of agent independence, durability and scalability can be added to Blum's criteria of being analyzable, publishable, secure, self-rehearsing, and humanly-usable, in making a password creation system. Its speed being faster and not relying on computation, the system is easier to use and more secure than earlier proposals [8]. We would be delighted to see this working demonstration leading to new families of more secure, usable password creation and retrieval systems. For example, the strong security constraints described in this paper 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 endeavor.

REFERENCES

1. 2014. Centrify Password Survey: Summary. https://www.centrify.com/resources/5778-centrify-password-survey-summary/. (2014). Accessed: 2017-12-20.
2. 2016. Password Managers: Single Point of Failure, or a Necessity for a Secure Enterprise? http://bedelsecurity.com/password-managers-single-point-of-failure-or-a-necessity-for-a-secure-enterprise-part-1/. (2016). Accessed: 2017-12-20.
3. 2017. The Equifax Data Breach: What to Do. https://www.consumer.ftc.gov/blog/2017/09/equifax-data-breach-what-do. (2017).
4. 2018. Amazon AWS S3 Cost Calculator. https://calculator.s3.amazonaws.com/index.html. (2018). Accessed: 2017-12-18.
5. D. Eastlake 3rd, J. Schiller, and S. Crocker. 2005. RFC4086: Randomness Requirements for Security. https://tools.ietf.org/html/rfc4086. (2005).
6. Jeremiah Blocki, Manuel Blum, and Anupam Datta. 2013. Naturally rehearsing passwords. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 361–380.
7. Jeremiah Blocki, Manuel Blum, Anupam Datta, and Santosh Vempala. 2014. Towards Human Computable Passwords. *arXiv preprint arXiv:1404.0024* (2014).
8. Manuel Blum and Santosh Srinivas Vempala. 2015. Publishable humanly usable secure password creation schemas. In *Third AAAI Conference on Human Computation and Crowdsourcing*.
9. Marc Brysbaert, Michaël Stevens, Paweł Mandera, and Emmanuel Keuleers. 2016. How Many Words Do We Know? Practical Estimates of Vocabulary Size Dependent on Word Definition, the Degree of Language Input and the Participant's Age. *Frontiers in Psychology* 7 (2016), 1116. DOI:http://dx.doi.org/10.3389/fpsyg.2016.01116
10. Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. 2014. The Tangled Web of Password Reuse.. In *NDSS*, Vol. 14. 23–26.
11. Shirley Gaw and Edward W. Felten. 2006. Password Management Strategies for Online Accounts. In *Proceedings of the Second Symposium on Usable Privacy and Security (SOUPS '06)*. ACM, New York, NY, USA, 44–55. DOI:http://dx.doi.org/10.1145/1143120.1143127
12. Le Quan Ha, E. I. Sicilia-Garcia, Ji Ming, and F. J. Smith. 2002. Extension of Zipf's Law to Words and Phrases. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1 (COLING '02)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 1–6. DOI: http://dx.doi.org/10.3115/1072228.1072345
13. George W Hartmann. 1946. Further evidence on the unexpected large size of recognition vocabularies among college students. *Journal of educational psychology* 37, 7 (1946), 436.

14. Nicholas J Hopper and Manuel Blum. 2001. Secure human identification protocols. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 52–66.

15. Ambarish Karole, Nitesh Saxena, and Nicolas Christin. 2011. A Comparative Usability Evaluation of Traditional Password Managers. In *Proceedings of the 13th International Conference on Information Security and Cryptology (ICISC'10)*. Springer-Verlag, Berlin, Heidelberg, 233–251. http://dl.acm.org/citation.cfm?id=2041036.2041056

16. Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. 2011. Of Passwords and People: Measuring the Effect of Password-composition Policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 2595–2604. DOI: http://dx.doi.org/10.1145/1978942.1979321

17. Kevan Lee. 2014. Four Methods to Create a Secure Password You'll Actually Remember. https://lifehacker.com/four-methods-to-create-a-secure-password-youll-actually-1601854240. (2014). Accessed: 2017-12-18.

18. Peter Lipa. 2016. The Security Risks of Using "Forgot My Password" to Manage Passwords. https://www.stickypassword.com/blog/the-security-risks-of-using-forgot-my-password-to-manage-passwords. (2016). Accessed: 2017-12-18.

19. W. Ma, J. Campbell, D. Tran, and D. Kleeman. 2010. Password Entropy and Password Quality. In *2010 Fourth International Conference on Network and System Security*. 583–587. DOI: http://dx.doi.org/10.1109/NSS.2010.18

20. George A Miller. 1956. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review* 63, 2 (1956), 81.

21. Peter Norvig. 2009. Natural language corpus data. (2009).

22. Richard Shay, Saranga Komanduri, Adam L. Durity, Phillip (Seyoung) Huh, Michelle L. Mazurek, Sean M. Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2014. Can Long Passwords Be Secure and Usable?. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2927–2936. DOI: http://dx.doi.org/10.1145/2556288.2557377

23. Hung-Min Sun, Yao-Hsin Chen, and Yue-Hsun Lin. 2012. oPass: A user authentication protocol resistant to password stealing and password reuse attacks. *IEEE Transactions on Information Forensics and Security* 7, 2 (2012), 651–663.

24. Aaron Toponce. 2010. Password Cards. https://pthree.org/2010/09/21/password-cards/. (2010). Accessed: 2017-12-18.

25. Aaron Toponce. 2011. Strong Passwords NEED Entropy. https://pthree.org/2011/03/07/strong-passwords-need-entropy/. (2011). Accessed: 2017-12-18.

26. Rick Wash, Emilee Rader, Ruthie Berman, and Zac Wellmer. 2016. Understanding Password Choices: How Frequently Entered Passwords Are Re-used across Websites. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. USENIX Association, Denver, CO, 175–188. https://www.usenix.org/conference/soups2016/technical-sessions/presentation/wash

APPENDIX

VARIANTS

Here are two variants of the algorithm found in section 2. The first is a higher-security one for the users who are good at mental arithmetic, which is more resistant to multiple plaintext attacks and easier to analyze. The second is a completely usable and adaptable one.

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

Algorithm 2: Higher Security Cue-Pin-Select

Data: Passphrase P of at least 6 random words
PIN K of 4 random digits
service name N
/* If the user struggles with numbers, the PIN can correspond to the length of the last 4 words */
Result: string S of around 12 characters
begin
  From N create string M of four characters
  /* This user-chosen string should be easy to remember */
  if *Service requires mixed-case* **then**
    M should be in mixed-case
  if *User had a previous cue for this service* **then**
    M[0] and M[2] become the next letters in the alphabet
  $L \longleftarrow$ Length(P)
  $V \longleftarrow 0$
  $S \longleftarrow$ ""
  for $i = 0$; $i < 4$; $i++$ **do**
    $X \longleftarrow M[i]$
    while $X \notin P$ **do**
      $X \longleftarrow$ letter following X in the alphabet
    $V \longleftarrow$ next occurrence of $X \in P$ after V
    /* Or the first occurrence if we reach the end of P */
    $V \longleftarrow V + K[i]$
    $P \longleftrightarrow$ Concatenate $(P[V], P[V+1], P[V+2])$
    if *M[i] is upper-case* **then**
      Make P upper-case
    $S \longleftarrow$ Concatenate (S, P)
  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* $L - y$ **then**
    $S \longleftarrow$ Suffix (S, y)
  /* We avoid the security measures by adding characters that don't change entropy, and remove the first few letters if needed */
  Print S

Algorithm 3: Adaptable Cue-Pin-Select