Password typo correction using discrete logarithms

Nikola K Blanchard

Password typo correction using discrete logarithms

Nikola K. Blanchard[1]

Digitrust, Loria, Université de Lorraine
Nikola.K.Blanchard@gmail.com, www.koliaza.com

Abstract. As passwords remain the main online authentication method, focus has shifted from naive entropy to how usability improvements can increase security. Chatterjee et al. recently introduced the first two typo-tolerant password checkers, which improve usability at no security cost but are technically complex. We look at the more general problem of computing an edit distance between two strings without having direct access to those strings — by storing the equivalent of a hash. We propose a simpler algorithm for this problem that is asymptotically quasi-optimal in both bits stored and exchanged, at the cost of more computation on the server.

Keywords: Usable security · Passwords · Discrete logarithm

1 Introduction

Despite recent advances in biometric authentication [12] and account linking [2], passwords are still the main method of authentication used online and will probably remain so in the near future. Countless studies have been written on the pitfalls of password-based authentication [11], with users creating bad passwords [4] or repeatedly dodging security measures [15,10]. Failing to login is increasingly frustrating, and forgetting one's password is now about as frustrating as forgetting one's keys [5]. To improve usability, some services like Facebook have discreetly adopted typo correction for the 2-3 most frequent typos, such as forgetting the caps lock or capitalising the first character on mobile [9].

In an innovative paper in 2016 [6], Chatterjee et al. discovered that authentication failures could turn 3% of the users away, but that a vast majority of errors comes from a few simple typos. They also developed a system called TypTop [7], which is efficient both computationally and memory-wise, and corrects up to 32% of typos. This system works by keeping a cache of allowed password hashes corresponding to the frequent typos made by the user, and updates this cache at each successful authentication. Those systems can actually have a positive impact on security as they make long passwords — which are more error-prone — much more usable, lowering the cost of using highly secure passwords.

We look at the general problem of storing information on the server that can allow typo correction while preventing an adversary in control of the server from computing the passwords from the stored information.

Main results. We introduce a metric called the *keyboard distance*, and a protocol to compute this distance (or the Hamming distance) between a queried string and a secret string, without it being possible to find the secret string in polynomial time (assuming the security of the discrete logarithm). This is non-trivial, as it was shown in [3] that any distance computation protocol can find the original password in a polynomial number of queries, which we prevent by having queries of non-uniform complexity.

2 Keyboard distance and algorithm

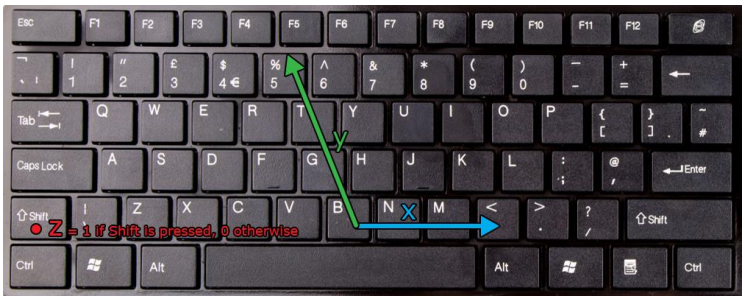

Fig. 1. Keyboard coordinate system, starting at the bottom left. The string "Arc" has coordinates $((1,1,1),(4,2,0),(3,0,0))$.

Before the algorithm, we must first introduce a distance between strings which, although simple, is not generally used. Let's consider a keyboard, with a standard QWERTY layout, as in Figure 1. The 48 main keys of the keyboard and the different characters they can create can easily be modelled by a 3-dimensional coordinate system. The first dimension corresponds to the horizontal position of the key (or the row), the second dimension to the vertical (the diagonal column), and the third dimension to the *modifiers*, here only considering Shift although it could easily be extended. This forms a subset of a $14 \times 4 \times 2$ lattice[1] as shown in Figure 1.

Definition 1. *Let s be a string of length n. The string coordinates of s are defined as the sequences $(x_i)_{1\leq i\leq n}$,$(y_i)_{1\leq i\leq n}$ and $(z_i)_{1\leq i\leq n}$, where (x_i, y_i, z_i) are the coordinates of the i-th letter in the previous coordinate system.*

Definition 2. *Let s and s' be strings of identical length n. Let the keyboard distance between s and s' be defined as the L^1-distance between their string coordinates: $d(s,s') = \sum_{1\leq i\leq n} (|x_i - x'_i| + |z_i - z'_i| + |z_i - z'_i|)$.*

[1] One could also add the space key, in which case the following proofs still work although with a slightly different structure. Similarly, adding the Alt key would only make it a 4-dimensional coordinate system.

By this definition, the distance between *homomorphic* and *homimorphic* is 1, but the distance between *homomorphic* and *Bomomorphic* is 3, the same as the distance between *homomorphic* and *homomor;jkc*.

The expected distance between two random n-character strings is then $\frac{59707}{10296} \times n$, or about 58 for 10-character keymashes.

Definition 3. *Let s be a string of length n, and let $(x_i)_{1\leq i\leq n}$,$(y_i)_{1\leq i\leq n}$ and $(z_i)_{1\leq i\leq n}$ be its string coordinates. Let p_i be the i-th prime number. We define the integral representation $X(s)$ of s as*

$$X(s) = \prod_{1\leq i\leq n} p_i^{x_i} \times p_{i+n}^{y_i} \times p_{i+2n}^{z_i}$$

To follow the example in the figure, the integral representation of "Arc" $2 \times 3^4 \times 5^3 \times 7 \times 11^2 \times 17 = 291579750$. The integral representation of "ArC" is $2 \times 3^4 \times 5^3 \times 7 \times 11^2 \times 17 \times 23 = 291579750 \times 23$.

We can now define the a cryptographic protocol to detect typos, inspired by the Diffie-Helman key exchange. Intuitively, we take a random element in a group and put it to the X-th power, where X is dependent on the password. Because of the function's structure, it is easy to compare the elements corresponding to two closely related strings. The security lies in the assumed hardness of computing the discrete logarithm.

Data: Username string U, Salt string S, Password string P
Group G, Pseudorandom number generator f
Result: An element $g_0 \in G$ as the "hashed" password sent to the server
begin
Compute the string coordinates $(x_i, y_i, z_i)_{1\leq i\leq |P|}$ of P
$X \longleftarrow \prod_{1\leq i\leq n} p_i^{x_i} \times p_{i+n}^{y_i} \times p_{i+2n}^{z_i}$; $Y \longleftarrow U + S$; $N \longleftarrow f(Y)$
Let g be a pseudorandom element g of G computed from N
Transfer $g_0 \longleftarrow g^X$ to the server

Algorithm 1: Key-setting/sending discrete logarithm algorithm

Remark 1. Alternatively, we could use a more intuitive definition, with $X(s) = \prod_{1\leq i\leq n} p_{3i-2}^{x_i} \times p_{3i-1}^{y_i} \times p_{3i}^{z_i}$. This way, strings that include others as prefixes have integral representations that are multiples of the prefixes' integral representations. As we only consider strings of constant length, this leads to higher values of $X(s)$ with no real advantage. On a standard keyboard, for a string s of length 10, $X(s) < 2^{966}$ with the second definition whereas $X(s) < 2^{768}$ with the first (and $X(s) < 2^{853}$ for length 12). In all cases, they are in expectation quite above 2^{250}, which is enough to prevent discrete logarithm attacks on small exponents [8].

Remark 2. The PRNG in the algorithm does not require a high level of security, and can simply be any algorithm to get an element from a set of pseudorandom bits — such as a PCG algorithm [13].

Data: Group G, constant D, maximum length n, nt $g_0, g_1 \in G$
Result: Keyboard distance between the passwords if it's less than D.
begin
  for i from 1 to D **do**
    for j from 0 to i **do**
      $L_0 \longleftarrow [\,]$
      $L_1 \longleftarrow [\,]$
      foreach $1 \leq a_1 \leq a_2 \leq ... \leq a_j \leq 3n$ **do**
        $X_0 \longleftarrow \prod_{a_k} p_{a_k}$
        $g' \longleftarrow g_0^{X_0}$
        $L_0 \longleftarrow$ Concatenate(L_0, g')
      foreach $1 \leq b_1 \leq b_2 \leq ... \leq b_{i-j} \leq 3n$ **do**
        $X_1 \longleftarrow \prod_{b_k} p_{b_k}$
        $g' \longleftarrow g_1^{X_1}$
        $L_1 \longleftarrow$ Concatenate(L_1, g')
      foreach $g' \in L_0$ **do**
        if $g' \in L_1$ **then**
          return i
  return REJECT

Algorithm 2: Distance-checking discrete logarithm algorithm

Remark 3. The reason why we compute two lists of elements is that computing errors where a_i is greater than expected is easy, as $g^{Xp_i} = (g^X)^{p_i}$. Computing errors the other way around is actually akin to computing a discrete logarithm in the group. As such, the distance computation in this algorithm always goes from the "smaller" to the "bigger" password, which can thankfully be mixed when the keyboard distance is greater than 1.

3 Security and performance

The security of this algorithm directly comes from the discrete logarithm assumption: computing P from g_0 corresponds exactly to solving the discrete logarithm with the promise that the solution is a $3n$-smooth number — for potentially high n in case of added padding. To implement it in practice, one would have to be careful to choose an appropriate group [1]. A cyclic group of order p with p a 2048-bit prime should be enough for now, and a similar algorithm could be adapted for elliptic curves.

With this framework, the login queries are all of the same format — a single element of the group. This could lead to a proof of optimality in terms of space and communication bits required, depending on the group used in practice. It also means that faking an id is not easier than the hardest typo-tolerant framework that accepts the same typos. As the size of the group is much greater than the general password space, the discrete logarithm assumption also implies that bruteforcing the password is the best avenue of attack.

Besides the fact that it only allows the correction of substitution errors, the main downside of this algorithm is the time needed to compute the distance.

This is still acceptable on the client side, where the main hurdle is squaring an element at most 1600 times in a large group. Using efficient libraries, this can be done in less than 10ms. However, the server-side computation is where the cost becomes prohibitive. For strings of length 12, checking whether they are at distance 1 takes at most 72 exponentiation operations, or less than 500 squaring operations, doable in a few ms. At distance 2, computation already takes 35 times more operations, which is on the edge of noticeable from the client-side. Checking whether they are at distance 3 (probably the highest reasonable distance for typos) is, alas, prohibitive, taking at least a few seconds. Using the trinomial revision, the number of expected exponentiations at distance $D \leq n$ is on average

$$\frac{1}{2}\sum_{i=0}^{D}\left(\binom{3n}{i}\binom{3n-i}{D-i}\right) = 2^{D-1} \times \sum_{i=0}^{D}\binom{3n}{D} \geq \frac{1}{2}\left(\frac{6n}{D}\right)^{D}.$$

The algorithm can also be adapted to compute Hamming distances, by checking all possible values for variants on a single letter instead of going by increasing keyboard distance.

4 Discussion

It was proved in [3] that black boxes that compute arbitrary distances between strings such as the one studied here are vulnerable to attacks with at most $poly(n)$ queries, and with $\theta(n)$ queries against the Hamming distance. The formula above illustrates why our method is not concerned by those lower bounds: although a linear number of queries would be enough to find the original string from the computed distances, most of those couldn't be computed because of their potentially exponential cost.

A second lower bound shown in [3] concerns the minimal number of communication and storage bits to obtain n bits of entropy, showing that in both cases, $n - o(n)$ bits are necessary. In our case, we store and send a single element of the group, and the security is that of a discrete logarithm attack against the group. We then have a quasi-linear time complexity for current commonly used groups, with real values currently corresponding to an overhead of a factor between 10 and 20.

Some questions remain, such as whether it is possible to obtain a linear storage or communication complexity (or whether stronger lower bounds are provable otherwise). Moreover, the typos corrected here only concern the Hamming or keyboard distances, and don't allow complex typos such as exchange of adjacent letters. It would be interesting to check whether the method could be expanded to more complex distance functions. Finally is also one potential risk that requires investigating with this method. The discrete logarithm assumption concerns normal elements of the group. However, the elements considered here are not random elements but X-th powers, with B-smooth X, for $101 \leq B \leq 181$. Although B-smooth numbers are essential in discrete logarithm problems [14], there seems to be no attack so far where X being B-smooth is an issue.

References

1. Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., Zimmermann, P.: Imperfect forward secrecy: How diffie-hellman fails in practice. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 5–17. CCS '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2810103.2813707
2. Batista, G.C., Miers, C.C., Koslovski, G.P., Pillon, M.A., Gonzalez, N.M., Simplicio, M.A.: Using Externals IdPs on OpenStack: A Security Analysis of OpenID Connect, Facebook Connect, and OpenStack Authentication. In: IEEE 32nd International Conference on Advanced Information Networking and Applications – AINA. vol. 00, pp. 920–927 (5 2018). https://doi.org/10.1109/AINA.2018.00135
3. Blanchard, N.K.: Usability: low tech, high security. Ph.D. thesis, Institut de Recherche en Informatique Fondamentale (2019)
4. Bonneau, J.: The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In: IEEE Symposium on Security and Privacy. pp. 538–552 (5 2012)
5. Centrify: Centrify password survey: Summary. Tech. rep., Centrify (2014), `https://www.centrify.com/resources/5778-centrify-password-survey-summary/`
6. Chatterjee, R., Athayle, A., Akhawe, D., Juels, A., Ristenpart, T.: pASSWORD tYPOS and how to correct them securely. In: IEEE Symposium on Security and Privacy. pp. 799–818. IEEE (2016)
7. Chatterjee, R., Woodage, J., Pnueli, Y., Chowdhury, A., Ristenpart, T.: The typtop system: Personalized typo-tolerant password checking. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 329–346. CCS '17, ACM, New York, NY, USA (2017)
8. Guillevic, A., Morain, F.: Discrete Logarithms. In: Mrabet, N.E., Joye, M. (eds.) Guide to pairing-based cryptography, p. 42. CRC Press - Taylor and Francis Group (Dec 2016), `https://hal.inria.fr/hal-01420485`
9. Lambert, P.: The case of case-insensitive passwords (6 2012), `https://web.archive.org/web/20190310221858/https://www.zdnet.com/article/the-case-of-case-insensitive-passwords/`
10. Lipa, P.: The security risks of using "forgot my password" to manage passwords (2016), `https://web.archive.org/web/20170802185615/https://www.stickypassword.com/blog/the-security-risks-of-using-forgot-my-password-to-manage-passwords/`
11. Ma, W., Campbell, J., Tran, D., Kleeman, D.: Password entropy and password quality. In: 4th International Conference on Network and System Security. pp. 583–587 (9 2010). https://doi.org/10.1109/NSS.2010.18
12. Memon, N.: How biometric authentication poses new challenges to our security and privacy [in the spotlight]. IEEE Signal Processing Magazine **34**(4), 196–194 (2017)
13. O'Neill, M.E.: PCG: A family of simple fast space-efficient statistically good algorithms for random number generation. ACM Transactions on Mathematical Software (2014)
14. Pomerance, C.: The role of smooth numbers in number theoretic algorithms. In: International Congress of Mathematicians. Citeseer (1994)
15. Ur, B., Noma, F., Bees, J., Segreti, S.M., Shay, R., Bauer, L., Christin, N., Cranor, L.F.: I added '!'at the end to make it secure": Observing password creation in the lab. In: Proceedings of the 11th symposium on usable privacy and security (2015)