Password Typo Correction Using Discrete Logarithms

Nikola K. Blanchard

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

Eighth international UBT Conference

October 26th, 2019

Password hashing is still an issue

Chatterjee et al. (2016 and 2017):

- Correcting typos improves usability
- It can be done without lowering security

Issues:

- Costly in terms of server computation
- Hard to implement

Previous contributions

Work done previously with Xavier Coquand and Ted Selker:

- Comparisons of server-side and client-side hashing
- Analysis of client-side hashing currently used
- Proposal of an efficient but complex password typo correction algorithm

This talk: generalised simpler algorithm for arbitrary typos.

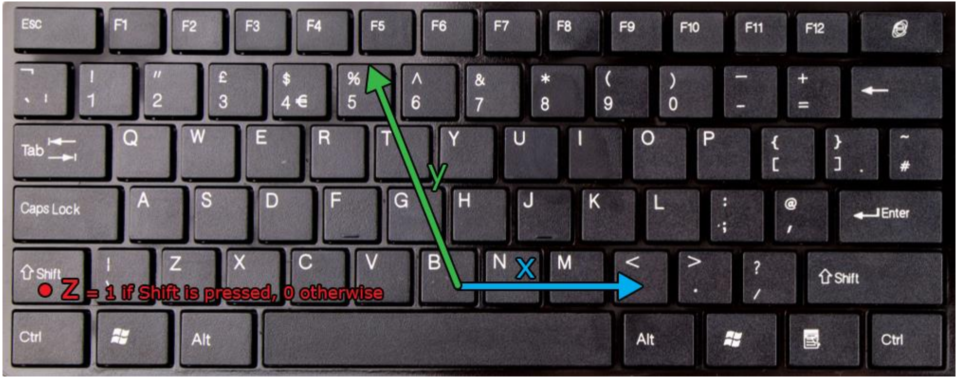
Computing string distances

General problem: computing string distances

Can we design an algorithm that stores $f(S)$ for a secret string S such that:

- For any S', we can efficiently compute $d(S, S')$
- It is not tractable to compute S' using only $f(S)$
- The storage and communication complexity are in $O(|S|)$

A new metric, the keyboard distance



Lower bounds: bad news

If we want to compute arbitrary distances:

- Can find initial string in $O(n)$ queries for many distances
- Extendable to $O(poly(n))$ for other distances
- Stays true even with probabilistic and inaccurate oracles

Can we still do something?

Discrete-log method

Hashing algorithm

Compute the string coordinates

$$(x_i, y_i, z_i)_{1 \leq i \leq |P|}$$

Compute the exponent

$$X \longleftarrow \prod_{1 \leq i \leq n} p_i^{x_i} \times p_{i+n}^{y_i} \times p_{i+2n}^{z_i}$$

For a random g, compute and send

$$g^X$$

Distance computing algorithm

begin
    for i from 1 to D **do**
        for j from 0 to i **do**
            $L_0 \longleftarrow [\,]$; $L_1 \longleftarrow [\,]$
            foreach $1 \leq a_1 \leq a_2 \leq \ldots \leq a_j \leq 3n$ **do**
                $X_0 \longleftarrow \prod_{a_k} p_{a_k}$; $g' \longleftarrow g_0^{X_0}$; $L_0 \longleftarrow \text{Concatenate}(L_0, g')$
            foreach $1 \leq b_1 \leq b_2 \leq \ldots \leq b_{i-j} \leq 3n$ **do**
                $X_1 \longleftarrow \prod_{b_k} p_{b_k}$; $g' \longleftarrow g_1^{X_1}$; $L_1 \longleftarrow \text{Concatenate}(L_1, g')$
            foreach $g' \in L_0$ **do**
                if $g' \in L_1$ **then**
                    return i
    return REJECT

Algorithm efficiency

Computing client costs:

- $\simeq 1600$ squaring operations, less than 10ms

Server costs exponential in the distance:

- ≤ 72 exponentiations $\simeq 500$ squaring operations at distance 1
- 35 times more at distance 2
- a few seconds at distance 3
- Not tractable for expected distance between two random strings (> 58)

Future work

A few open questions:

- Can this kind of algorithm be extended to other distances?
- Can the efficiency be improved to usable levels?
- Which groups are best suited?
- Can we obtain linear communication complexity?
- Is using an X-th power for 200-smooth X problematic?

Thank you for your attention