

Master Key Algorithm

Authors: Abdulaziz Al-Zoman
Raed Al-Fayez
Abdulrahman ALGhadir

Published: 20/Sept./ 2010

This document explains:

- The idea behind this algorithm.
- The model which is based on.
- The uses of it.

1. The Model of Master Key Algorithm

The Master Key Algorithm is an algorithm to group all possible variants for certain string into a unique key which groups them under it, consider these two strings ABC and ABF where A, B, C and F are code points (code point is a character which follow certain languages) for certain languages also C and F are code points which they have similarity thus ABC and ABF will have one unique key generated by this algorithm.

Master Key Algorithm consists of these Elements:

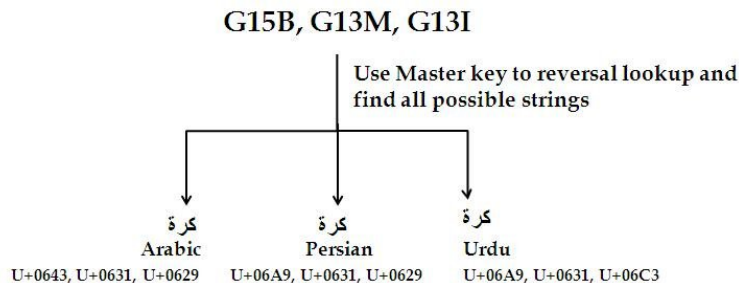
- 1-**Languages Table (LT)**: is a lookup table that defines the code points where they belong for certain language.
- 2-**Variant Table (VT)**: is a table which records all relations for certain language where these relations represent the similarity of code points for that language with other code points from the same script.
- 3-**Combined Variant Table (CVT)**: is a table which contains all VT(s) of certain script(s) into one single table.
- 4-**Group Variant Table (GVT)**: is the final lookup table which used to generate the master key where each entry of it represents a GVT key and list of relations that follow that key.

This algorithm has two main functions:

- Generating master key.

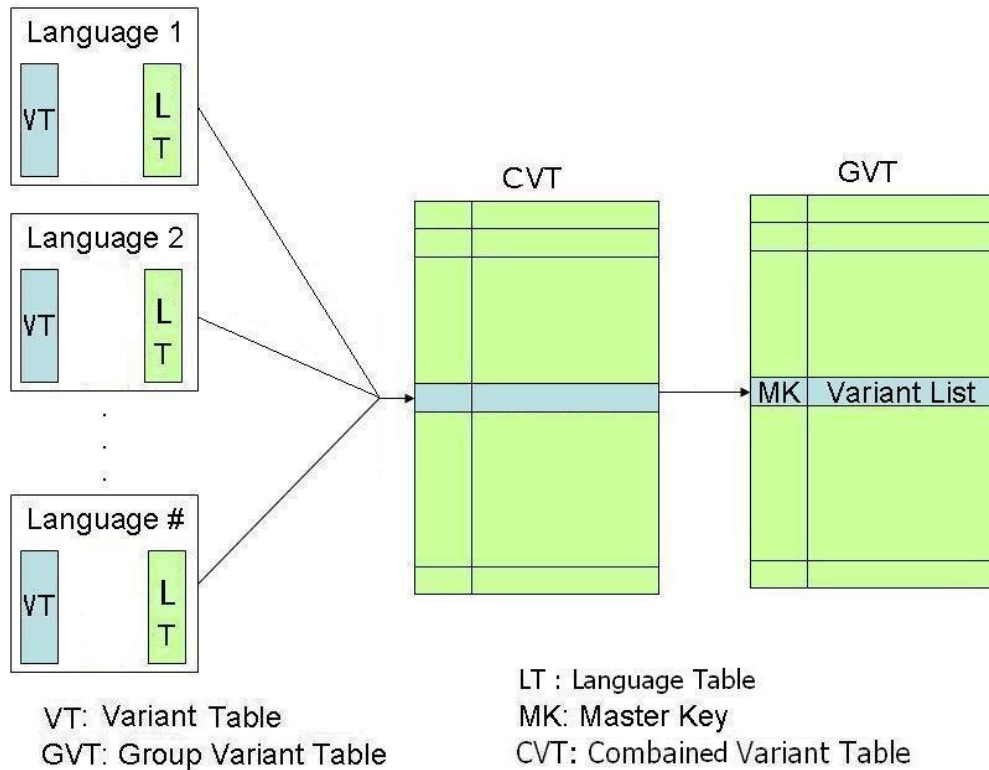


- Finding all similar strings for certain master key.



Usages of the algorithm:

Mainly this algorithm was designed to solve problems with similar domain names but this algorithm can be used on different fields where they have common problems same as domain names.



1.1 Language Table (LT)

A Language Table (LT) consists of a set of code points (a subset taken from Unicode table) representing the allowed characters for a supported language by a registry to write an IDN under a given TLD. These code points will be referred as base characters. An LT table will be used to validate the input string (label) to make sure that it consists of characters from the language. Also, it will be used to generate valid variants for a certain string (or Master Key).

E.g. assume that we have a languages which contains four code points :

أ, ب, ج and د

Now our LT will have

(U+0623), (U+0628), (U+062C) and (U+062F) these are Unicode's for ا, ب, ج and د respectively

1.2 Variant Tables (VT)

A Variant table is constructed by human (preferably linguistic experts of the language who also understand Unicode) who can illustrate the similarity relations between the characters in a LT (which the variant table is built for) and other characters from the whole script table.

A similarity relation between characters is defined as either Exact or Typo. An Exact relation is when the similarity between the concerned characters is identical(as mirror) , while a Typo relation is defined when the concerned characters look alike but not identical (typo/style match).

A VT consists of records, each record contains: a base character (from LT), a list of variants characters indicating where are the positions of similarity [**B**eginning (B), **M**edial (M), **F**inal(F), **I**solated(I)] and what type of relation: **E**xact (E) or **T**ypo(T). It has the following form:

`<base_char_hex>; [<target_char_hex>”(“<pos>:<rel>”)” (“,” | ε)]*`

Where:

<base_char_hex>: a code point of a base character .

<target_char_hex>: a code point for a character from the same script which has a relation with the base .

<pos>: the position of similarity relationship. It can be one or more of the basic shapes: **B**eginning, **M**edial, **F**inal and **I**solated.

<rel>: the type of similarity between the <base_char_hex> and <target_char_hex> either TTypo (T) or Exact (E).

Example:

0641; 06A7 (FI: T), 06A7 (BM: E)

In this example the code point (ف) U+0641 has a typo relation with the code point (ف) U+06A7 in the Final and Isolated positions and it has an exact relation with same code point (ف) U+06A7 in the Beginning and Medial positions.

1.3 Combined Variant Table (CVT)

As an initial phase to create the registry's Group Variant Table, a Combined Variant Table (CVT) first created that combines all VTs into one table. Hence, a CVT table keeps a list of all relations across the script for the supported characters. Since the CVT is used to build the GVT thus storing CVT is not necessary. A CVT table has the same structure as a VT table. The Base characters of the CVT table are the supper set of all characters from the LTs.

$$\text{BaseChar}(\text{CVT}) = \bigcup_{i=0}^n \text{BaseChar}(\text{VT}_i)$$

Where n= number of VTs

1.3.1 Building CVT

Building CVT involves merging all VTs so that all relations for a certain base character will be grouped together.

The process of building CVT

CVT ← an empty array

For every VT do the following

For every **BaseChar** in VT do the following

Collect all variants (as pairs of Code point, Position, Relation) for the **BaseChar** from all VTs into one group

Remove the **BaseChar** (along with its pairs) from other VTs

Normalize the group by:

Removing duplicates pairs

Merging pairs that have same code point and relation by taking union of their positions^[1]

For ever pairs that have same code point with different relations do:

Amend the positions for the Typo relation to exclude the Exact positions^[2]

Delete pairs with no positions

Add the resulting group to CVT

[1] Assume we have this set of relations for character 0064 (B, BM, M, BMF) all of them have Exact relation the highest position will be BMF.

[2] Assume we have this set of relations for character 0064 (B, BM, M, BMFI) all of them have Typo relation find the highest position which is BMFI from step [2] the difference between these two relations BMF and BMFI is I assign this new relation with Typo relation and add it.

1.4 Group Variant Table

Group Variant Table is a table which groups, all characters which they have similarity relations with each other. Every group of characters will have a unique key. Every character will appear in only one key in GVT

A GVT table consists of a list of records (different from VT records). Each record starts with a unique key which identifies a group of all characters that have similarity relations in particular position. The record format as follows:

```
<KEY>; [<char_hex> "]"("<pos>":Q)" ("," | ε)]* [<char_hex1>"&"<char_hex2>"&" "("<pos>:<rel>")" ("," | ε)]*
```

Regular expression for GVT line

Where:

<KEY>: is a GVT key in the form "G"# ("B","M","F", "I").

< char_hex>: a code point of a query character one or more used when we want to generate the Master Key.

< char_hex1> and < char_hex2>: code points of two characters that have similarity relationship.

<pos>: the character position where the similarity exists.. It is indicated by one or more of the following letters: B (Beginning), M (Medial), F (Final) and I (Isolated).

<rel>: the type of similarity relationship between < char_hex1>: and < char_hex2>. It is indicated by either T (Typo), E (Exact).

1.4.1 Building GVT

The CVT table will be used to build GVT. The goal here is to group all characters that are similar in any given position into one key and record their relations.

The process of building GVT

RS ← an empty queue for characters that Request to be searched

GVT ← an empty table

done ← an empty list used to record all characters which have been processed

For every **BaseChar** in CVT do the following

 Add **BaseChar** to the empty queue

 While queue is not empty do the following

 Get the first element from the queue if the element hasn't been processed do:

 Add the element as Query relation ^[1]

 Get all relations for this element from CVT and record them^[2]

 Add all code points of the relations to the queue

 Find all **BaseChar** where the element shows as part of its relations

 Add the relations where the element shows up and remove these relations

 Add the code point of **BaseChar** in queue where element showed up in it

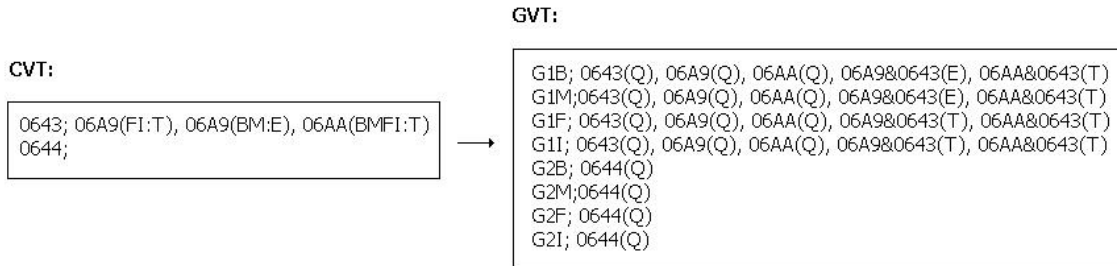
 Flag the element as processed

 Add the results of last step on GVT with unique key corresponded to the position^[3]

[1] Master key is in form <code>(<pos>:Q)

[2] Relation is in form <code1>&<code2>(<pos>:T|E)

[3] GVT key will be in form "G"<num>("B"|"M"|"F"|"I")

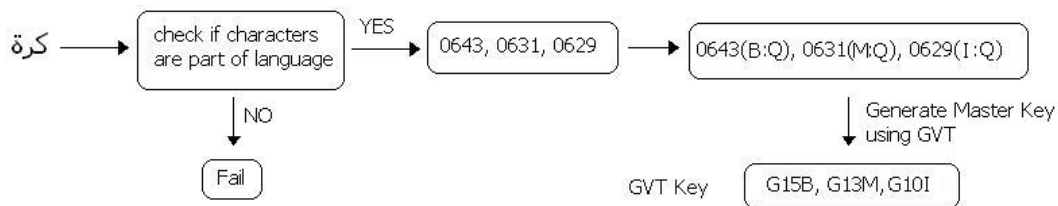


Notice: every key of type <code>(Q) is unique and it will only appear once in only one particular GVT key.

1.2 Generating Master Key (Group Variant Key)

Let's assume we want to generate a GVT key for an Arabic word (which means that every character will have four glyphs at max) the process will be like this:

- 1) Make sure that required word is from same language which means no mixing between languages is allowed. (this can be done by checking every character from word with its VT which belongs to it)
- 2) Now convert the word into hex form.
- 3) Scan every character in the word and see what its position depends on it appearance.
- 4) Now we will have a sequence of hex and its positions in form:
 <code1>(<pos1>:Q) <code2>(<pos2>:Q) <code#>(<pos#>:Q)
- 5) Take every code and check where does it appear in GVT table and get the GVT key which refers to it.
- 6) Done!



1.5 Regenerating GVT

Regenerating GVT needed when a new language (VT) in same script required to be added to the existence GVT of that script. There are two scenarios: first regenerating GVT will not result any merging of old GVT keys, second if regenerating will result merging of old keys.

Scenario one:

Merging will be possible in this case and the process will be like this:

- 1) Generate GVT for the new language.
- 2) Take every key in new language GVT and see where the master keys appear in old GVT.
- 3) If one of master keys from new language GVT key appears in any key of the old GVT take the variant list of the key in new language GVT and add them at the key in old GVT where the master key appeared.
- 4) The rest of keys in new language GVT append them at the end old GVT.

Scenario two:

Regenerating will result merging of old GVT keys which will result disorder of existence old GVT keys.

In this case only solution is to regenerate the old GVT from scratch with the new language VT then regenerate every GVT key from new GVT list.

1.6 Finding all variant strings

Once we have a GVT key for certain string it is possible to find all variant combinations from that key. Simple approach is to find all characters which have “Q” relation type for GVT key, once this done every GVT key will have one or more Master character in the list. Now find all permutations for all masters which you found.