# 1 Introduction

RTI2 is a variable length bit packed rainbow table format developed by www.freerainbowtables.com. It reduces disk usage through the variable length data chains, though each data file has one consistent data chainrow length, and through the use of prefix indexes. It is in some ways similar to the RTI format but designed to be easier to modify items in subsequent minor versions and this flexibility adds a fair amount of complexity. Additionally, it is designed so that a transition away from the file naming for information may be possible as this will be necessary for more complex table sets in the future.

The details of the format itself are placed in the public domain so that all applications may have compatible implementations. All source code is under the GPL 2.0 or GPL 3.0. Any closed source implementation ***MUST*** not be based on any of the source code unless explicit notices regarding public domain licensing of code snippets are declared as will be the case for any code contained in this document. Information obtained through the forums on www.freerainbowtables.com shall be considered public domain unless notated otherwise. If you are developing a closed source implementation please be cautious about downloading any attachments that may contain source code from the forums that may fall under a version of the GPL.

All *.rti, *.rti.index, and *.rti2 data files are freely and publicly available for download and unlimited license. Any distribution of the tables themselves is freely permitted as long as the following conditions are met:
1) no fee besides the cost of the storage device(s) is charged
2) it is clearly indicated that the tables came from www.freerainbowtables.com

Please note that this documentation refers to RTI 2.0 and subsequent minor versions may have changes. The on disk format follows and afterwards a longer explanation of the format will hopefully make the details clear. For any clarifications do not hesitate to ask in the forums on www.freerainbowtables.com or email admin@freerainbowtables.com

Also note that in RTI 2.0 due to bugs the character set and keyspace encoding should be read but should not be used. Instead use the from the file name as you would for RTI or RT formats. This problem will be fixed in RTI 2.1.

Where the algorithm is Half LM Challenge, the salt is assumed to be 0x1122334455667788 and mistakenly is not encoded as described.

# 2 RTI 2.0 File Format

## 2.1 Header

- Tag "RTI2" (4 bytes)
- Minor "0" (1 byte)
- Start point bits "SP" (1 byte)
  - The number of bits for the start point in section 2.3
- End point bits "EP" (1 byte)
  - The number of bits for the end point, not in the prefix index, in section 2.3
- Check point bits "CP" (1 byte)
  - The number of bits for check points in section 2.3. If this value is 0 then no data is stored for Check point positions
- File index (4 bytes)
  - The data file for the table starting with 0 for the first file
- Files (4 bytes)
  - the total number of data files that make up the table
- Rainbow table parameters
  - Minimum start point (8 bytes)
    - a starting offset for all start points in the table
  - Chain length (4 bytes)
  - Table index (4 bytes)
  - Algorithm (1 byte)

        - 0 - Reserved
        - 1 - LM
        - 2 - NTLM
        - 3 - MD2
        - 4 - MD4
        - 5 - MD5
        - 6 - Double MD5
        - 7 - Binary Double MD5
        - 8 - Cisco Pix (96 bit MD5)
        - 9 - SHA1
        - 10 - MySQL SHA1(double binary sha1)
        - 11 - SHA256
        - 12 - SHA384
        - 13 - SHA512
        - 14 - RipeMD160
        - 15 - MSCache
        - 16 - Half LM Challenge
        - 17 - Second Half LM Challenge
        - 18 - NTLM Challenge
        - 19 - Oracle
    - Reduction function (1 byte)
        - 0 - RC ("RainbowCrack" uses divide - project-rainbowcrack.com)
        - 1 - FPM ("Fixed Point Multiplication" - tobtu.com)
        - 2 - GRT ("GPU RT" uses lookup tables -cryptohaze.com)
    - If the Algorithm contains a salt
        - Salt length (1 byte)
        - Salt (Salt length bytes)
            - Stored in binary. For Half LM Challenge the salt is "\x11\x22\x33\x44\x55\x66\x77\x88"
    - Number of sub key spaces (1 byte)
    - Sub key space
        - Number of hybrid character sets (1 byte)
        - Hybrid character set Note: These are for 1, 2, 3, 4 byte characters
            - Password length (1 byte)
            - Character set flags (1 - MultiByteChar1 set, 2 - MultiByteChar2 set, 4 - MultiByteChar3 set, 8 - Multi-ByteChar4 set) (1 byte)
            - MultiByteChar1 set size (MultiByteChar1 set size = 1 * (Data + 1)) (1 byte if MultiByteChar1 set flag else 0 bytes "MultiByteChar1 set size = 0")
            - MultiByteChar1 set (1 byte characters) (MultiByteChar1 set size bytes)
            - MultiByteChar2 set size (MultiByteChar2 set size = 2 * (Data + 1)) (1 byte if MultiByteChar2 set flag else 0 bytes "MultiByteChar2 set size = 0")
            - MultiByteChar2 set (2 byte characters) (MultiByteChar2 set size bytes)
            - MultiByteChar3 set size (MultiByteChar3 set size = 3 * (Data + 1)) (1 byte if MultiByteChar3 set flag else 0 bytes "MultiByteChar3 set size = 0")
            - MultiByteChar3 set (3 byte characters) (MultiByteChar3 set size bytes)
            - MultiByteChar4 set size (MultiByteChar4 set size = 4 * (Data + 1)) (1 byte if MultiByteChar4 set flag else 0 bytes "MultiByteChar4 set size = 0")
            - MultiByteChar4 set (4 byte characters) (MultiByteChar4 set size bytes)
- Check point positions (4*CP bytes)
    - The position in the chain where check point bits are stored if CP is not 0

### 2.2 Index

- First prefix (8 bytes)
- Number of prefix indexes (4 bytes)
- Prefix indexes
    - Prefix chain count (1 byte)

### 2.3 Data

- Chains (ceiling[[SP+CP+EP]/8] bytes)
    - End point (EP bits)
    - Start point (SP bits)
    - Check points (CP bits)
    - Padding bits (8*ceiling[(SP+CP+EP)/8]-SP-CP-EP bits)

## 3 Additional implementation information

Sub key spaces are designed to allow us to do per position tables which includes our hybrid2 format. Each sub key space will contain at least 1 hybrid set.

A hybrid set within the sub key space consists of one of the following:
MultiByteChar1 set containing 1 byte characters (ASCII),
MultiByteChar2 set containing 2 bytes characters (UTF8),
3, or 4 byte UTF8 character encodings.
Combining hybrid sets at a password position allows a way to order the character sets and to add a few arbitrary characters when constructing a table such as a few common UTF8 characters in German or other languages. The password length is the position in the string starting at 1.

An example encoding of "numeric#1-4"

- "\x04" Sub key spaces (1 byte)
- Sub key space
    - "\x01" Number of hybrid sets (1 byte)
    - Hybrid set
        - "\x01" Password length (1 byte)
        - "\x01" Character set flags (1 - MultiByteChar1 set, 2 - MultiByteChar2 set, 4 - MultiByteChar3 set, 8 - MultiByteChar4 set) (1 byte)
        - "\x09" MultiByteChar1 set size (MultiByteChar1 set size = 1 * (Data + 1)) (1 byte if MultiByteChar1 set flag else 0 bytes "MultiByteChar1 set size = 0")
        - "0123456789" MultiByteChar1 set (1 byte characters) (10 set size bytes)
- Sub key space
    - "\x01" Number of hybrid sets (1 byte)
    - Hybrid set
        - "\x02" Password length (1 byte)
        - "\x01" Character set flags (1 - MultiByteChar1 set, 2 - MultiByteChar2 set, 4 - MultiByteChar3 set, 8 - MultiByteChar4 set) (1 byte)
        - "\x09" MultiByteChar1 set size (MultiByteChar1 set size = 1 * (Data + 1)) (1 byte if MultiByteChar1 set flag else 0 bytes "MultiByteChar1 set size = 0")

        - "0123456789" MultiByteChar1 set (1 byte characters) (10 set size bytes)

- Sub key space
    - "\x01" Number of hybrid sets (1 byte)
    - Hybrid set
        - "\x03" Password length (1 byte)
        - "\x01" Character set flags (1 - MultiByteChar1 set, 2 - MultiByteChar2 set, 4 - MultiByteChar3 set, 8 - MultiByteChar4 set) (1 byte)
        - "\x09" MultiByteChar1 set size (MultiByteChar1 set size = 1 * (Data + 1)) (1 byte if MultiByteChar1 set flag else 0 bytes "MultiByteChar1 set size = 0")
        - "0123456789" MultiByteChar1 set (1 byte characters) (10 set size bytes)

- Sub key space
    - "\x01" Number of hybrid sets (1 byte)
    - Hybrid set
        - "\x04" Password length (1 byte)
        - "\x01" Character set flags (1 - MultiByteChar1 set, 2 - MultiByteChar2 set, 4 - MultiByteChar3 set, 8 - MultiByteChar4 set) (1 byte)
        - "\x09" MultiByteChar1 set size (MultiByteChar1 set size = 1 * (Data + 1)) (1 byte if MultiByteChar1 set flag else 0 bytes "MultiByteChar1 set size = 0")
        - "0123456789" MultiByteChar1 set (1 byte characters) (10 set size bytes)

**Prefix Index details**

Each prefix chain count on disk is the number of chains in the prefix index. Example of populating and reading the prefix indexes:
let count be the number of prefix indexes

```
uint8_t *indexTmp;
std::vector<uint32> prefixIndex;
int a;
int sum = 0;

indexTmp = new uint8_t[count];

fin.read( (char*) indexTmp, count )

prefixIndex.reserve(count + 1);
prefixIndex.push_back(sum);

for (a = 0; a < count; a++)
{
  sum += indexTmp[a];
  prefixIndex.push_back(sum);
}

delete [] indexTmp;
```

**Data reading example**
This is just a straight forward in memory conversion to RT for a single data chain.

```
uint64_t chainrow = 0x000000D9B02B763A;
uint8_t startPointBits = 33;
uint8_t endPointBits = 15;
uint64_t firstPrefix = 1;
uint64_t minimumStartPoint = 36500549;
uint32_t chainPrefixOffset = 32781377;  // the value from prefixIndex[n]
uint64_t ep,sp;

uint64_t endPointMask = (( (uint64_t) 1 ) << endPointBits ) - 1;
uint64_t startPointMask = (( (uint64_t) 1 ) << startPointBits ) - 1;
uint32_t startPointShift = endPointBits;
uint32_t chainSize = (startPointBits + endPointBits + 7) >> 3;

ep = (firstPrefix + chainPrefixOffset - 1 ) << endPointBits;
ep |= chainrow & endPointMask;
sp = (( chainrow >> startPointShift ) & startPointMask ) + minimumStartPoint;
```

Multi-byte character set encoding example

UTF-8 character set "aÀ‰"
a U+0061, UTF-8: 61
À U+00C0, UTF-8: C380
‰U+2030, UTF-8: E280B0
...

- Hybrid character set
  - "\x01" Password length (1 byte)
  - "\x07" Character set flags (1 - MultiByteChar1 set, 2 - MultiByteChar2 set, 4 - MultiByteChar3 set, 8 - Multi-ByteChar4 set) (1 byte)
  - "\x00" MultiByteChar1 set size (MultiByteChar1 set size = 1 * (Data + 1)) (1 byte if MultiByteChar1 set flag else 0 bytes "MultiByteChar1 set size = 0")
  - "a" MultiByteChar1 set (1 byte characters) (MultiByteChar1 set size bytes)
  - "\x00" MultiByteChar2 set size (MultiByteChar2 set size = 2 * (Data + 1)) (1 byte if MultiByteChar2 set flag else 0 bytes "MultiByteChar2 set size = 0")
  - "\xc3\x80" MultiByteChar2 set (2 byte characters) (MultiByteChar2 set size bytes)
  - "\x00" MultiByteChar3 set size (MultiByteChar3 set size = 3 * (Data + 1)) (1 byte if MultiByteChar3 set flag else 0 bytes "MultiByteChar3 set size = 0")
  - "\xe2\x80\xb0" MultiByteChar3 set (3 byte characters) (MultiByteChar3 set size bytes)
- "\x03" - Sub key spaces
  - "\x04" Number of hybrid character sets
    - "\x01" "\x01" "\x19" "ABCDEFGHIJKLMNOPQRSTUVWXYZ" - Password length, Character set flag, MultiByteChar1 set size, MultiByteChar1 set
    - "\x05" "\x01" "\x19" "abcdefghijklmnopqrstuvwxyz" - Password length, Character set flag, MultiByteChar1 set size, MultiByteChar1 set
    - "\x02" "\x01" "\x23" "abcdefghijklmnopqrstuvwxyz0123456789" - Password length, Character set flag, MultiByteChar1 set size, MultiByteChar1 set
    - "\x01" "\x01" "\x09" "0123456789" - Password length, Character set flag, MultiByteChar1 set size, Multi-ByteChar1 set

- "\x04" Number of hybrid character sets
  - "\x01" "\x01" "\x19" "ABCDEFGHIJKLMNOPQRSTUVWXYZ" - Password length, Character set flag, MultiByteChar1 set size, MultiByteChar1 set
  - "\x05" "\x01" "\x19" "abcdefghijklmnopqrstuvwxyz" - Password length, Character set flag, MultiByteChar1 set size, MultiByteChar1 set
  - "\x02" "\x01" "\x23" "abcdefghijklmnopqrstuvwxyz0123456789" - Password length, Character set flag, MultiByteChar1 set size, MultiByteChar1 set
  - "\x02" "\x01" "\x09" "0123456789" - Password length, Character set flag, MultiByteChar1 set size, Multi-ByteChar1 set
- "\x04" Number of hybrid character sets
  - "\x01" "\x01" "\x19" "ABCDEFGHIJKLMNOPQRSTUVWXYZ" - Password length, Character set flag, MultiByteChar1 set size, MultiByteChar1 set
  - "\x05" "\x01" "\x19" "abcdefghijklmnopqrstuvwxyz" - Password length, Character set flag, MultiByteChar1 set size, MultiByteChar1 set
  - "\x02" "\x01" "\x23" "abcdefghijklmnopqrstuvwxyz0123456789" - Password length, Character set flag, MultiByteChar1 set size, MultiByteChar1 set
  - "\x03" "\x01" "\x09" "0123456789" - Password length, Character set flag, MultiByteChar1 set size, Multi-ByteChar1 set