Luhn algorithm

From Wikipedia, the free encyclopedia

The **Luhn algorithm** or **Luhn formula**, also known as the "modulus 10" or "mod 10" algorithm, is a simple checksum formula used to validate a variety of identification numbers, such as credit card numbers, IMEI numbers, National Provider Identifier numbers in the US, and Canadian Social Insurance Numbers. It was created by IBM scientist Hans Peter Luhn and described in U.S. Patent No. 2,950,048, filed on January 6, 1954, and granted on August 23, 1960.

The algorithm is in the public domain and is in wide use today. It is specified in ISO/IEC 7812-1.[1] It is not intended to be a cryptographically secure hash function; it was designed to protect against accidental errors, not malicious attacks. Most credit cards and many government identification numbers use the algorithm as a simple method of distinguishing valid numbers from mistyped or otherwise incorrect numbers.

Contents [hide]

Description [edit]

The formula verifies a number against its included check digit, which is usually appended to a partial account number to generate the full account number. This number must pass the following test:

1. From the rightmost digit, which is the check digit, moving left, double the value of every second digit; if the product of this doubling operation is greater than 9 (e.g., 8 × 2 = 16), then sum the digits of the products (e.g., 16: 1 + 6 = 7, 18: 1 + 8 = 9).
2. Take the sum of all the digits.
3. If the total modulo 10 is equal to 0 (if the total ends in zero) then the number is valid according to the Luhn formula; else it is not valid.

Assume an example of an account number "7992739871" that will have a check digit added, making it of the form 7992739871x:

Account number	7	9	9	2	7	3	9	8	7	1	**x**
Double every other	7	**18**	9	**4**	7	**6**	9	**16**	7	**2**	**x**
Sum digits	7	**9**	9	4	7	6	9	**7**	7	2	**x**

The sum of all the digits in the third row is **67+x**.

The check digit (x) is obtained by computing the sum of the non-check digits then computing 9 times that value modulo 10 (in equation form, (67 × 9 mod 10)). In algorithm form:

1. Compute the sum of the non-check digits (67).
2. Multiply by 9 (603).
3. The last digit, 3, is the check digit. Thus, **x=3**.

(Alternative method) The check digit (x) is obtained by computing the sum of the other digits then subtracting the units digit from 10 (67 => Units digit 7; 10 − 7 = check digit 3). In algorithm form:

1. Compute the sum of the digits (67).
2. Take the units digit (7).
3. Subtract the units digit from 10.
4. The result (3) is the check digit. In case the sum of digits ends in 0, 0 is the check digit.

This, makes the full account number read 79927398713.

Each of the numbers 79927398710, 79927398711, 79927398712, 79927398713, 79927398714, 79927398715, 79927398716, 79927398717, 79927398718, 79927398719 can be validated as follows.

1. Double every second digit, from the rightmost: (1×2) = 2, (8×2) = 16, (3×2) = 6, (2×2) = 4, (9×2) = 18
2. Sum all the *individual* digits (digits in parentheses are the products from Step 1): x (the check digit) + (2) + 7 + (1+6) + 9 + (6) + 7 + (4) + 9 + (1+8) + 7 = x + 67.
3. If the sum is a multiple of 10, the account number is possibly valid. Note that **3** is the only valid digit that produces a sum (70) that is a multiple of 10.
4. Thus these account numbers are all invalid except possibly 79927398713 which has the correct check digit.

Alternately (if you don't want to confuse yourself by performing an algorithm on the whole number including the checksum digit), you can use the same checksum creation algorithm (mentioned a couple paragraphs up), ignoring the checksum already in place, as if it had not yet been calculated, and now you were calculating it for the first time. Then calculate the checksum and compare this calculated checksum to the original checksum included with the credit card number. If the included checksum matches the calculated checksum, then the number is valid.

Strengths and weaknesses [edit]

The Luhn algorithm will detect any single-digit error, as well as almost all transpositions of adjacent digits. It will not, however, detect transposition of the two-digit sequence *09* to *90* (or vice versa). It will detect 7 of the 10 possible twin errors (it will not detect *22* ↔ *55*, *33* ↔ *66* or *44* ↔ *77*).

Other, more complex check-digit algorithms (such as the Verhoeff algorithm and the Damm algorithm) can detect more transcription errors. The Luhn mod N algorithm is an extension that supports non-numerical strings.

Because the algorithm operates on the digits in a right-to-left manner and zero digits affect the result only if they cause shift in position, zero-padding the beginning of a string of numbers does not affect the calculation. Therefore, systems that pad to a specific number of digits (by converting 1234 to 0001234 for instance) can perform Luhn validation before or after the padding and achieve the same result.

Prepending a 0 to odd-length numbers enables you to process the number from left to right rather than right to left, doubling the odd-place digits.

The algorithm appeared in a US Patent[2] for a hand-held, mechanical device for computing the checksum. It was therefore required to be rather simple. The device took the mod 10 sum by mechanical means. The *substitution digits*, that is, the results of the double and reduce procedure, were not produced mechanically. Rather, the digits were marked in their permuted order on the body of the machine.

Implementation of standard Mod 10 [edit]

The implementations below are in Python.

Verification of the check digit [edit]

```
def luhn_checksum(card_number):
    def digits_of(n):
        return [int(d) for d in str(n)]
    digits = digits_of(card_number)
    odd_digits = digits[-1::-2]
    even_digits = digits[-2::-2]

    checksum = sum(odd_digits)
    for d in even_digits:
        checksum += sum(digits_of(d*2))
    return checksum % 10

def is_luhn_valid(card_number):
    return luhn_checksum(card_number) == 0
```

Calculation of the check digit [edit]

The algorithm above checks the validity of an input with a check digit. Calculating the check digit requires only a slight adaptation of the algorithm—namely:

1. Append a zero check digit to the partial number and calculate checksum
2. If the (sum mod 10) == 0, then the check digit is 0
3. Else, the check digit = 10 - (sum mod 10)

```
def calculate_luhn(partial_card_number):
    check_digit = luhn_checksum(int(partial_card_number) * 10)
    return check_digit if check_digit == 0 else 10 - check_digit
```

See also [edit]

- Bank card number

References [edit]

1. ^ ISO/IEC 7812-1:2006 Identification cards – Identification of issuers – Part 1: Numbering system
2. ^ US Patent 2,950,048 - *Computer for Verifying Numbers*, Hans P Luhn, August 23, 1960

External links [edit]

- Implementation in 88 languages on the Rosetta Code project
- Open Source implementation in PowerShell
- Luhn implementations in JavaScript
- Validation of Luhn in PHP
- Implementation in C
- Ruby: Luhn validation, Luhn generation
- Luhn validation class in C#
- Luhn validation implementation in Java
- Luhn validation in SQL
- Luhn algorythms for non-numeric characters

Categories: Modular arithmetic | Checksum algorithms | Error detection and correction