# Parallel

Steve Thomas (Steve at tobtu dot com)

# Specification

## Symbols/functions

```
**                  Exponential
%                   Modulus
||                  Concatenate two strings
^                   Xor two strings
SHA512              Calculates a SHA512 of a string
truncate(string, N) Truncates a string to N bytes
zeros(N)            Creates an N byte string of zeros
```

## Password hash

Inputs:

```
string  password
string  salt
integer t_cost
integer outlen
```

Algorithm:

```
t_cost_1          = t_cost % 2 ** 16
t_cost_2          = floor(t_cost / 2 ** 16)
t_cost_sequential = floor(2 ** floor((t_cost_2 - 1) / 2) * (3 - (t_cost_2 % 2)))
                  = 1, 2, 3, 4, 6, 8, 12, ... (Note floor(-1 / 2) = -1)
t_cost_ parallel  = floor(2 ** floor((t_cost_2 - 1) / 2) * (3 - (t_cost_2 % 2)))
                  = 1, 2, 3, 4, 6, 8, 12, ... (Note floor(-1 / 2) = -1)


key = SHA512(SHA512(salt) || password))

// Work
for i = 0 to t_cost_sequential - 1
      // Clear work
      work = zeros(64)

      for j = 0 to t_cost_ parallel
            work = work ^ SHA512(BIG_ENDIAN_64(i) || BIG_ENDIAN_64(j) || key)

      // Finish
      key = SHA512(SHA512(work || key))
      key = truncate(key, outlen) || zeros(64 - outlen)

return truncate(key, outlen)
```

**KDF**

Inputs:

```
string  password
string  salt
integer t_cost
integer outlen
```

Algorithm:

```
t_cost_ parallel = floor(2 ** floor((t_cost - 1) / 2) * (3 - (t_cost % 2)))
                 = 1, 2, 3, 4, 6, 8, 12, ... (Note floor(-1 / 2) = -1)


key  = SHA512(SHA512(salt) || password))
work = zeros(64)

// Work
for j = 0 to t_cost_ parallel
       work = work ^ SHA512(BIG_ENDIAN_64(i) || BIG_ENDIAN_64(j) || key)

// Finish
i = 0
while length(out) < outlen
       out = out || SHA512(BIG_ENDIAN_64(i) || work || password)
       i = i + 1
return truncate(out, outlen)
```

**Statement**

There are no deliberately hidden weaknesses (backdoor, etc.).

# Initial Security Analysis

This is best for low memory applications or when FPGAs or GPUs are present. It's very simple and is as resistant to collisions as the underling hash function.

# Efficiency Analysis

The attacker-defender ratio is 1. Any advancements in cracking are advancements for the defender. If ASICs come out that can crack this hash they more than likely can be used by the defender.

# Intellectual Property Statement

The scheme is and will remain available worldwide on a royalty free basis, and I am unaware of any patent or patent application that covers the use or implementation of the submitted algorithm.