# Some Notes on Impementation of the Comb Method for Elliptic Curves in the mbedtls Library

We are interested in the file `ecp.c` (see `https://github.com/Mbed-TLS/mbedtls/tree/development/library`).

1. The function

   ```
   static void ecp_comb_recode_core( unsigned char x[],
                                     size_t d,
                                     unsigned char w,
                                     const mbedtls_mpi *m )
   ```

   recodes scalar $*m$ into the array $x[]$ of binary representations of consecutive columns in the multilayer construction build by the comb method.

2. Argument $d$ is the initial width of each layer (i.e., before recoding). After recoding the width of each layer is $d + 1$. So initially $d$ correspons to $a$ from the description of the comb algorithm. In mbedtls we have $b = a$, so we have only one vertical subblock.

3. Argument $w$ is the height of the construction (i.e., $h$ from the algorithm description).

4. The function `ecp_comb_recode_scalar` ensures that the scalar $m$ is an odd number[1] (cf. the parity trick). The parity trick is reverted in the function `ecp_mul_comb_after_precomp`.

5. The first loop present in `ecp_comb_recode_core` is just reading the bits of $m$ that are $d$ bits away from yeach other. They compose $x[k] = I_{j,k}$, where $I_{j,k}$ is from the description of the algorithm, and where $j \in \{0\}$, $k = 0, \ldots, d-1$ in the mbedtls.

6. The aim of the second loop is to set all the bits to '1' in the least significant layer ( referred as $e_0$ in the lecture ).

7. Since the input scalar to the `ecp_comb_recode_core` is an odd number, we know that the least significant bit (lsb) of $m$ has value 1, hence the second loop starts with $i = 1$.

8. Note that by setting '1's in the least significant layer we:

[1] for simplicity we neglect the fact that $m$ is a pointer to te scalar, so we should use notation $*m$ to denote the scalar itself

(a) reduce the set of possible variables in each 1-bit-width column (so for each $i$ we reduce the set of possible values $x[i]$), thus for given $w$ we reduce storage requirements for the array $T[\,]$ of precomputed points; as a result we may increase value of $w$ making the whole construction narrower,

(b) make the column a non-zero one, that is after recoding each column $x[i]$ will contain a non-zero value (this is important in preventing side channel attacks).

9. The price for reduction of memory requirements is extension to $d+1$ of the width of the construction (cf. `memset( x, 0, d+1 )` and the limit $i \leq d$ for the second loop in the `ecp_comb_recode_core` ).

10. In $x[i]$ the most significant bit encodes the sign (the bit answers the question: should we add the point taken from $T[\,]$, or should we substract it?), and the bits with indices 6-0 encode the absolute value in the column, that is the index of $T[\,]$ under which the appropriate point is stored. The point from $T[\,]$ is read by the `ecp_select_comb` function.

11. In the second loop of the function `ecp_comb_recode_core` :

```
    for( i = 1; i <= d; i++ )
    {
        /* Add carry and update it */
        cc   = x[i] & c;
        x[i] = x[i] ^ c;
        c = cc;

        /* Adjust if needed, avoiding branches */
        adjust = 1 - ( x[i] & 0x01 );
        c   |= x[i] & ( x[i-1] * adjust );
        x[i] = x[i] ^ ( x[i-1] * adjust );
        x[i-1] |= adjust << 7;
    }
```

the layers can be treated as binary representations of $w$ 'separate' numbers. Each value $x[i]$ is a snapshot of a single bit in each of these numbers (the snapshot is taken on the same, $i$th position in each number). Consequently, $x[i]$ can be treated as binary vector of these single bits.

12. We know that `x[i-1] & 0x01 == 1` (for `i==1` this is true, for the next '$i$'s: on the basis of induction).

13. If `x[i] & 0x01 == 0` then we sum up two binary snapshots $x[i] + x[i-1]$ of bits in the $w$ 'separate' numbers. The result is to be stored in $x[i]$, and in this way we set the least significant bit in $x[i]$ to 1 (that is we flip the bit on position $i$ in the least significant layer from 0 to 1).

To not change the value of the sum of points from $T[\,]$ accumulated in the fial result $R$ we have to set sign to '$-$' on the position corresponding to $x[i-1]$ (see the line 12 above). Why change of the sign does not change the value of the sum is explained below.

14. The addition of the snaphots $x[i] + x[i-1]$ in the binary expansion of the layers is done gradually:
    (a) In the line 10 we detect the bits that are equal to 1 on the same positions in vectors $x[i]$, $x[i-1]$, so we need preserve them for the carry.
    (b) The `xor` executed in line 11 correspons to binary addition of the vectors.

15. We have to remember that finally operations are to be made on the points precomputed and stored in the array $T[\,]$. Namely, we are going to accumulate in `R` the sum:

$$T[x[0]] + 2^1 \cdot T[x[1]] + 2^2 \cdot T[x[2]] + \dots$$

16. Note that

$$x[i] = (m_{d\cdot(w-1)+i} m_{d\cdot(w-2)+i} \dots m_{d+i} m_i)_2,$$

where $m_j$ is the $j$th bit of $m$, and

$$T[x[i]] = \Big(2^{d\cdot(w-1)} \cdot m_{d\cdot(w-1)+i} + 2^{d\cdot(w-2)} \cdot m_{d\cdot(w-2)+i} + \dots + 2^d \cdot m_{d+i} + 2^0 \cdot m_i\Big) \cdot G,$$

where $G$ is the basepoint. So the addition $x[i] + x[i-1]$ with the change of sign of the point from the array $T[\,]$ that corresponds to $x[i-1]$, translates to the following:

$$\begin{aligned}
&2^i \cdot T[x[i] + x[i-1]] + (-2^{i-1} \cdot T[x[i-1]]) = \\
&= 2^i \cdot \Big(2^{d\cdot(w-1)} \cdot (m_{d\cdot(w-1)+i} + m_{d\cdot(w-1)+(i-1)}) + 2^{d\cdot(w-2)} \cdot (m_{d\cdot(w-2)+i} + (m_{d\cdot(w-2)+(i-1)})) + \\
&\qquad + 2^d \cdot (m_{d+i} + m_{d+(i-1)}) + 2^0 \cdot (m_i + m_{i-1})\Big) \cdot G - \\
&- 2^{i-1} \cdot \Big(2^{d\cdot(w-1)} \cdot m_{d\cdot(w-1)+(i-1)} + 2^{d\cdot(w-2)} \cdot m_{d\cdot(w-2)+(i-1)} + \dots + 2^d \cdot m_{d+(i-1)} + 2^0 \cdot m_{i-1}\Big) \cdot G \\
&= 2^i \cdot \Big(2^{d\cdot(w-1)} \cdot m_{d\cdot(w-1)+i} + 2^{d\cdot(w-2)} \cdot m_{d\cdot(w-2)+i} + \dots + 2^d \cdot m_{d+i} + 2^0 \cdot m_i\Big) \cdot G + \\
&+ (2^i - 2^{i-1}) \cdot \Big(2^{d\cdot(w-1)} \cdot m_{d\cdot(w-1)+(i-1)} + 2^{d\cdot(w-2)} \cdot m_{d\cdot(w-2)+(i-1)} + \dots + 2^d \cdot m_{d+(i-1)} + 2^0 \cdot m_{i-1}\Big) \cdot G \\
&= 2^i \cdot T[x[i]] + (2 \cdot 2^{i-1} - 2^{i-1}) \cdot T[x[i-1]] = 2^i \cdot T[x[i]] + 2^{i-1} \cdot T[x[i-1]]
\end{aligned}$$