RIPEMD-160 Gadget Implementation using the `halo2` API

Saravanan Vijayakumaran

April 3, 2024

1 Motivation

The HASH160 opcode in the Bitcoin protocol hashes a given byte sequence with the SHA-256 hash function followed by the RIPEMD-160 hash function. It is used to obtain 20-byte hashes of objects like public keys and Bitcoin scripts.

To prove the validity of Bitcoin response scripts for Pay-to-Public-Key (P2PK) and Pay-to-Script-Hash (P2SH) UTXOs using the `halo2` API, we need an implementation of a HASH160 gadget. A SHA-256 gadget implementation already exists in the `zcash/halo2` repository. In this document, we describe the design and implementation of a RIPEMD-160 gadget. It is heavily inspired by SHA-256 gadget implementation.

2 RIPEMD-160 Specification

A specification of the RIPEMD-160 hash function is available at `https://homes.esat.kuleuven.be/~bosselae/ripemd160.html`, along with a C implementation. We repeat it here for convenience and for fixing the notation used to describe the gadget implementation.

The RIPEMD-160 hash function operates on 32-bit words. The RIPEMD-160 round function processes 16 message words (64 bytes) at a time. So the input is padded to have a length in bytes which is a multiple of 64. *We assume that the input to the RIPEMD-160 gadget has already been padded correctly.*

Padding Scheme RIPEMD-160 uses the same padding scheme as MD4 [1]. Suppose that the input is a b-bit message where $b \geq 0$.

- A single 1 bit is appended to the message followed by a sequence of 0 bits until the length of the padded message in bits is congruent to 448 modulo 512.
- A 64-bit representation of the length b is then appended to get a bit sequence whose length is a multiple of 512. The 4 lower order bytes of b are appended first in little-endian (LE) order (least significant byte appears first), followed by the 4 higher order bytes of b also in LE order. For example, the 64-bit representation of $b = 24$ is `0x1A00 0000 0000 0000`.

Functions The following functions are used in RIPEMD-160.

$$\begin{aligned}
f_1(x, y, z) &= x \oplus y \oplus z, \\
f_2(x, y, z) &= (x \wedge y) \vee (\neg x \wedge z)\,, \\
f_3(x, y, z) &= (x \vee \neg y) \oplus z, \\
f_4(x, y, z) &= (x \wedge z) \vee (y \wedge \neg z)\,, \\
f_5(x, y, z) &= x \oplus (y \vee \neg z)\,.
\end{aligned}$$

The variables x, y, z are 32-bit words. The symbols $\oplus$, $\wedge$, $\vee$, and $\neg$ respectively represent the bitwise XOR, AND, OR, and NEGATE operations.

Phase Constants The RIPEMD-160 round function has two independent halves whose outputs are combined at the end. Each half has 80 rounds which are divided into 5 phases of 16 rounds each. The following constants are used in each phase where K_i is used in the ith round of the left half and K'_i is used in the ith round of the right half.

$$\begin{aligned}
K_1 &= \texttt{0x00000000}, & K'_1 &= \texttt{0x50A28BE6}, \\
K_2 &= \texttt{0x5A827999}, & K'_2 &= \texttt{0x5C4DD124}, \\
K_3 &= \texttt{0x6ED9EBA1}, & K'_3 &= \texttt{0x6D703EF3}, \\
K_4 &= \texttt{0x8F1BBCDC}, & K'_4 &= \texttt{0x7A6D76E9}, \\
K_5 &= \texttt{0xA953FD4E}, & K'_5 &= \texttt{0x00000000}.
\end{aligned}$$

The non-zero values above are derived from the square roots and cube roots of 2, 3, 5, and 7.

Message Word Selection The input to the round function is a 16-word message block. In each of the 80 rounds, one of the 16 message words is used. For a round index ranging from 0 to 79 and message index ranging from 0 to 15, the round index to message index mapping for the *left* half is as follows.

$$\begin{aligned}
r[0..80] = [&0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, \\
&7, 4, 13, 1, 10, 6, 15, 3, 12, 0, 9, 5, 2, 14, 11, 8, \\
&3, 10, 14, 4, 9, 15, 8, 1, 2, 7, 0, 6, 13, 11, 5, 12, \\
&1, 9, 11, 10, 0, 8, 12, 4, 13, 3, 7, 15, 14, 5, 6, 2, \\
&4, 0, 5, 9, 7, 12, 2, 10, 14, 1, 3, 8, 11, 6, 15, 13]
\end{aligned}$$

The round index to message index mapping for the *right* half is as follows.

$$\begin{aligned}
r'[0..80] = [&5, 14, 7, 0, 9, 2, 11, 4, 13, 6, 15, 8, 1, 10, 3, 12, \\
&6, 11, 3, 7, 0, 13, 5, 10, 14, 15, 8, 12, 4, 9, 1, 2, \\
&15, 5, 1, 3, 7, 14, 6, 9, 11, 8, 12, 2, 10, 0, 4, 13, \\
&8, 6, 4, 1, 3, 11, 15, 0, 5, 12, 2, 13, 9, 7, 10, 14, \\
&12, 15, 10, 4, 1, 5, 8, 7, 6, 2, 13, 14, 0, 3, 9, 11]
\end{aligned}$$

Rotate Left Amount Each of the 80 rounds of both halves involves a rotate left operation on a 32-bit word. The amount of rotation ranges from 5 to 15. The round index to rotate amount mapping for the *left* half is as follows.

$$\begin{aligned} s[0..80] = [&11,14,15,12,5,8,7,9,11,13,14,15,6,7,9,8,\\ &7,6,8,13,11,9,7,15,7,12,15,9,11,7,13,12,\\ &11,13,6,7,14,9,13,15,14,8,13,6,5,12,7,5,\\ &11,12,14,15,14,15,9,8,9,14,5,6,8,6,5,12,\\ &9,15,5,11,6,8,13,12,5,12,13,14,11,8,5,6] \end{aligned}$$

The round index to rotate amount mapping for the *right* half is as follows.

$$\begin{aligned} s'[0..80] = [&8,9,9,11,13,15,15,5,7,7,8,11,14,14,12,6,\\ &9,13,15,7,12,8,9,11,7,7,12,7,6,15,13,11,\\ &9,7,15,11,8,6,6,14,12,13,5,14,13,13,7,5,\\ &15,5,8,11,14,14,6,14,6,9,12,9,12,5,15,8,\\ &8,5,12,9,12,5,14,6,8,13,6,5,15,13,11,11] \end{aligned}$$

Compression Function Let $\boxplus$ denote addition modulo 2^{32} and $\texttt{rol}_s$ denote the rotate left operation by s positions. RIPEMD-160 uses a 5-word chaining variable whose initial value is given as follows.

$h_0 = \texttt{0x67452301}$, $h_1 = \texttt{0xEFCDAB89}$, $h_2 = \texttt{0x98BADCFE}$, $h_3 = \texttt{0x10325476}$, $h_4 = \texttt{0xC3D2E1F0}$.

Suppose that the input after padding consists of t 16-word blocks. Let $X_i[j]$ denote the message word at index j in the ith block, where $0 \le i \le t-1$ and $0 \le j \le 15$. The following pseudocode describes the RIPEMD-160 compression function.

Algorithm 1 RIPEMD-160 compression function

```
for $i = 0$ to $t-1$ do
    $A \leftarrow h_0$; $B \leftarrow h_1$; $C \leftarrow h_2$; $D \leftarrow h_3$; $E \leftarrow h_4$
    $A' \leftarrow h_0$; $B' \leftarrow h_1$; $C' \leftarrow h_2$; $D' \leftarrow h_3$; $E' \leftarrow h_4$
    for $j = 0$ to $79$ do
        $p = \left\lfloor \frac{j}{16} \right\rfloor + 1$                                  ▷ Calculate phase index
        $T \leftarrow \texttt{rol}_{s[j]}\,(A \boxplus f_p(B,C,D) \boxplus X_i[r[j]] \boxplus K[p]) \boxplus E$
        $A \leftarrow E$; $E \leftarrow D$; $D \leftarrow \texttt{rol}_{10}(C)$; $C \leftarrow B$; $B \leftarrow T$
        $p' = 6 - p$
        $T \leftarrow \texttt{rol}_{s'[j]}\,(A' \boxplus f_{p'}(B',C',D') \boxplus X_i[r'[j]] \boxplus K'[p]) \boxplus E'$
        $A' \leftarrow E'$; $E' \leftarrow D'$; $D' \leftarrow \texttt{rol}_{10}(C')$; $C' \leftarrow B'$; $B' \leftarrow T$
    end for
    $T \leftarrow h_1 \boxplus C \boxplus D'$; $h_1 \leftarrow h_2 \boxplus D \boxplus E'$; $h_2 = h_3 \boxplus E \boxplus A'$;
    $h_3 \leftarrow h_4 \boxplus A \boxplus B'$; $h_4 \leftarrow h_0 \boxplus B \boxplus C'$; $h_0 \leftarrow T$.
end for
```

The variables A, B, C, D, E represent the state of the left half of the compression function and the variables A', B', C', D', E' represent those of the right half. The final hash value is present in the 5 words h_0, h_1, h_2, h_3, h_4. To convert these words into the hash byte sequence, we write each word in the sequence $h_0, h_1, \ldots, h_4$ as 4 bytes in LE order.

3 Gates

We need to implement gates for the following operations.

- The functions $f_1, f_2, \ldots, f_5$.
- Rotate left operation $\texttt{rol}_s$ for shift amount $s \in \{5, 6, \ldots, 15\}$.
- Addition of four words modulo 2^{32} to calculate $S = A \boxplus f_p(B, C, D) \boxplus X_i[r[j]] \boxplus K[j]$
- Addition of two words modulo 2^{32} to calculate $\texttt{rol}_s(S) \boxplus E$
- Addition of three words modulo 2^{32} to calculate expressions of the form $h_1 \boxplus C \boxplus D'$

All our gate implementations use ideas from the SHA-256 Table16 gate implementations[1]. We use six columns $a_0, a_1, \ldots, a_5$, out of which the first three correspond to the 16-bit lookup table and the last three are general advice columns.

a_0	a_1	a_2
tag(R)	R	spread(R)

In each row, the lookup table checks that the values in a_0, a_1, a_2 satisfy the following constraints.

- The value in a_1 is a 16-bit value R.
- The value spread(R) in a_2 is a 32-bit value that equals the spread version of R, i.e. it is obtained by inserting 0 bits in the odd positions of R.
- The value tag(R) in a_0 depends on R as follows:

$$\text{tag}(R) = \begin{cases} 0 & \text{if } 0 \leq R < 2^8, \\ 1 & \text{if } 2^8 \leq R < 2^9, \\ 2 & \text{if } 2^9 \leq R < 2^{10}, \\ 3 & \text{if } 2^{10} \leq R < 2^{11}, \\ 4 & \text{if } 2^{11} \leq R < 2^{12}, \\ 5 & \text{if } 2^{12} \leq R < 2^{13}, \\ 6 & \text{if } 2^{13} \leq R < 2^{14}, \\ 7 & \text{if } 2^{14} \leq R < 2^{15}, \\ 8 & \text{if } 2^{15} \leq R < 2^{16}, \end{cases}$$

The value of tag(R) is used to constrain R to have a certain bit length. If tag(R) $= 0$, then R is atmost an 8-bit integer. If tag(R) ≤ 1, then R is atmost a 9-bit integer and so on.

3.1 Function Gates

Among the functions $f_1, f_2, \ldots, f_5$, we need to implement gates only for the first three, as $f_4(x, y, z) = f_2(z, x, y)$ and $f_5(x, y, z) = f_3(y, z, x)$.

[1] `https://zcash.github.io/halo2/design/gadgets/sha256/table16.html`

3.1.1 f_1 gate

Recall that $f_1(B,C,D) = B \oplus C \oplus D$ where B, C, D are 32-bit words. We assume that the spread forms of the input words are available from previous operations. They might be specifically calculated for input into f_1 or available as outputs of other gates. The layout of the gate is as shown below, where `s_f1` is the selector that activates it.

`s_f1`	a_0	a_1	a_2	a_3	a_4	a_5
1		R_0^{even}	spread(R_0^{even})	spread(B^{lo})	spread(C^{lo})	spread(D^{lo})
		R_0^{odd}	spread(R_0^{odd})	spread(B^{hi})	spread(C^{hi})	spread(D^{hi})
		R_1^{even}	spread(R_1^{even})			
		R_1^{odd}	spread(R_1^{odd})			

The following constraint is enforced by the gate.

$$\begin{aligned}
&\text{spread}(R_0^{even}) + 2 \cdot \text{spread}(R_0^{odd}) + 2^{32}\text{spread}(R_1^{even}) + 2^{33}\text{spread}(R_1^{odd}) \\
&= \text{spread}(B^{lo}) + \text{spread}(C^{lo}) + \text{spread}(D^{lo}) + 2^{32}\left[\text{spread}(B^{hi}) + \text{spread}(C^{hi}) + \text{spread}(D^{hi})\right]
\end{aligned}$$

The values in R_0^{even}, R_1^{even} contain the output, i.e. the low and high half-words of $f_1(B,C,D)$.

3.1.2 f_2 gate

Recall that $f_2(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$ where B, C, D are 32-bit words. This function also appears in the SHA-256 compression function. Our gate implementation is the same as the one in the `halo2` SHA-256 gadget, with the exception that we compute and OR the results of $B \wedge C$ and $\neg B \wedge D$ in a single gate. In the SHA-256 gadget, the OR calculation is deferred and rolled up into a subsequent addition operation.

As before, we assume that the spread forms of the input words are available from previous operations. The layout of the gate is as shown below, where `s_f2f4` is the selector that activates it. The name of the selector reflects the fact that the same layout can be used for f_4.

`s_f2f4`	a_0	a_1	a_2	a_3	a_4	a_5
1		P_0^{even}	spread(P_0^{even})	spread(B^{lo})	spread(C^{lo})	
		P_0^{odd}	spread(P_0^{odd})	spread(B^{hi})	spread(C^{hi})	
		P_1^{even}	spread(P_1^{even})			
		P_1^{odd}	spread(P_1^{odd})			
		Q_0^{even}	spread(Q_0^{even})		spread(D^{lo})	spread($\neg B^{lo}$)
		Q_0^{odd}	spread(Q_0^{odd})		spread(D^{hi})	spread($\neg B^{hi}$)
		Q_1^{even}	spread(Q_1^{even})	sum^{lo}	carry	
		Q_1^{odd}	spread(Q_1^{odd})	sum^{hi}		

The following constraints are enforced by the gate.

$$\begin{aligned}
&\text{spread}(P_0^{even}) + 2 \cdot \text{spread}(P_0^{odd}) + 2^{32}\text{spread}(P_1^{even}) + 2^{33}\text{spread}(P_1^{odd}) \\
&\qquad = \text{spread}(B^{lo}) + \text{spread}(C^{lo}) + 2^{32}\left[\text{spread}(B^{hi}) + \text{spread}(C^{hi})\right], \\
&\text{spread}(2^{16}-1) = \text{spread}(B^{lo}) + \text{spread}(\neg B^{lo}), \\
&\text{spread}(2^{16}-1) = \text{spread}(B^{hi}) + \text{spread}(\neg B^{hi}), \\
&\text{spread}(Q_0^{even}) + 2 \cdot \text{spread}(Q_0^{odd}) + 2^{32}\text{spread}(Q_1^{even}) + 2^{33}\text{spread}(Q_1^{odd}) \\
&\qquad = \text{spread}(\neg B^{lo}) + \text{spread}(D^{lo}) + 2^{32}\left[\text{spread}(\neg B^{hi}) + \text{spread}(D^{hi})\right], \\
&sum^{lo} + 2^{16} sum^{hi} + 2^{32}\text{carry} = P_0^{odd} + \cdot Q_0^{odd} + 2^{16}\left[P_1^{odd} + Q_1^{odd}\right], \\
&\text{carry} = 0.
\end{aligned}$$

The top four rows calculate $B \wedge C$ which is present in the half-words P_0^{odd}, P_1^{odd}. The bottom four rows calculate $\neg B \wedge D$ which is present in the half-words Q_0^{odd}, Q_1^{odd}. The values in sum^{lo}, sum^{hi} contain the output, i.e. the low and high half-words of $f_2(B, C, D)$.

The values of spread$(\neg B^{lo})$ and spread$(\neg B^{hi})$ are obtained by constraining their respective sums with spread(B^{lo}) and spread(B^{hi}) to be `0x55555555` = spread$(2^{16} - 1)$. Since atmost one of the bits at the same locations in $B \wedge C$ and $\neg B \wedge D$ can be 1, the sum $P_0^{odd} + \cdot Q_0^{odd} + 2^{16}\left[P_1^{odd} + Q_1^{odd}\right]$ has zero carry and equals $(B \wedge C) \vee (\neg B \wedge D)$.

3.1.3 f_3 gate

Recall that $f_3(B, C, D) = (B \vee \neg C) \oplus D$ where B, C, D are 32-bit words. As before, we assume that the spread forms of the input words are available from previous operations. The layout of the gate is as shown below, where `s_f3f5` is the selector that activates it. The name of the selector reflects the fact that the same layout can be used for f_5.

`s_f3f5`	a_0	a_1	a_2	a_3	a_4	a_5
1		sum_0^{even}	spread(sum_0^{even})	spread$(\neg C^{lo})$	spread(B^{lo})	spread(C^{lo})
		sum_0^{odd}	spread(sum_0^{odd})	spread$(\neg C^{hi})$	spread(B^{hi})	spread(C^{hi})
		sum_1^{even}	spread(sum_1^{even})			
		sum_1^{odd}	spread(sum_1^{odd})			
		or^{lo}	spread(or^{lo})	spread(D^{lo})		
		or^{hi}	spread(or^{hi})	spread(D^{hi})		
		R_0^{even}	spread(R_0^{even})			
		R_0^{odd}	spread(R_0^{odd})			
		R_1^{even}	spread(R_1^{even})			
		R_1^{odd}	spread(R_1^{odd})			

The following constraints are enforced by the gate.

$$
\begin{aligned}
&\text{spread}(2^{16} - 1) = \text{spread}(C^{lo}) + \text{spread}(\neg C^{lo}),\\
&\text{spread}(2^{16} - 1) = \text{spread}(C^{hi}) + \text{spread}(\neg C^{hi}),\\
&\text{spread}(sum_0^{even}) + 2 \cdot \text{spread}(sum_0^{odd}) + 2^{32}\text{spread}(sum_1^{even}) + 2^{33}\text{spread}(sum_1^{odd})\\
&\qquad\qquad = \text{spread}(B^{lo}) + \text{spread}(\neg C^{lo}) + 2^{32}\left[\text{spread}(B^{hi}) + \text{spread}(\neg C^{hi})\right],\\
&\text{spread}(or^{lo}) + 2^{32} \cdot \text{spread}(or^{hi})\\
&\qquad = \text{spread}(sum_0^{even}) + \cdot\text{spread}(sum_0^{odd}) + 2^{32}\left[\text{spread}(sum_1^{even}) + \text{spread}(sum_1^{odd})\right],\\
&\text{spread}(R_0^{even}) + 2 \cdot \text{spread}(R_0^{odd}) + 2^{32}\text{spread}(R_1^{even}) + 2^{33}\text{spread}(R_1^{odd})\\
&\qquad = \text{spread}(or^{lo}) + \text{spread}(D^{lo}) + 2^{32}\left[\text{spread}(or^{hi}) + \text{spread}(D^{hi})\right].
\end{aligned}
$$

The top four rows calculate $\neg C$ and $B + \neg C$. The even bits of $B + \neg C$ are present in the quarter-words $sum_0^{even}, sum_1^{even}$. Similarly, the odd bits of $B + \neg C$ are present in the quarter-words sum_0^{odd}, sum_1^{odd}.

To calculate $B \vee \neg C$, we make use of the following observation: Suppose the sum $b + c$ of two bits b, c is given by the pair of bits sum_{odd}, sum_{even}, where sum_{odd} is the carry bit. Then $b \vee c$ equals $sum_{odd} + sum_{even}$.

The bottom four rows are used to calculate the XOR of $B \vee \neg C$ and D. The values in R_0^{even}, R_1^{even} contain the output, i.e. the low and high half-words of $f_3(B, C, D)$.

3.2 Rotate Left Gates

We need to verify the calculation of $\texttt{rol}_s(W)$ where W is a 32-bit word and s ranges from 5 upto and including 15. We use a size 11 array of selectors `s_rotate_left` to activate each gate.

3.2.1 $\texttt{rol}_5$ gate

We decompose $W = w_{31}w_{30}\cdots w_2w_1w_0$ into four parts $a^{hi}(3), a^{lo}(2), b(11), c(16)$,

$$\underbrace{w_{31}\cdots w_{29}}_{a^{hi}(3)}\underbrace{w_{28}w_{27}}_{a^{lo}(2)}\underbrace{w_{26}w_{25}w_{24}\cdots w_{16}}_{b(11)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)},$$

where the numbers in parentheses indicate lengths of the respective parts in bits. The rotated word in terms of these four parts is given by

$$\texttt{rol}_5(W) = \underbrace{w_{26}w_{25}w_{24}\cdots w_{16}}_{b(11)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)}\underbrace{w_{31}\cdots w_{29}}_{a^{hi}(3)}\underbrace{w_{28}w_{27}}_{a^{lo}(2)}.$$

The layout of the gate is as shown below, where `s_rotate_left[0]` is the selector that activates it.

`s_rotate_left[0]`	a_0	a_1	a_2	a_3	a_4	a_5
1	tag($b(11)$)	$b(11)$		$a^{lo}(2)$	W^{lo}	$\texttt{rol}_5(W)^{lo}$
		$c(16)$		$a^{hi}(3)$	W^{hi}	$\texttt{rol}_5(W)^{hi}$

The following constraints are enforced by the gate where $\texttt{range\_check}(x, j, k) = \prod_{i=j}^{k}(x - i)$.

$$\begin{aligned}
W^{lo} + 2^{16}W^{hi} &= c(16) + 2^{16}b(11) + 2^{27}a^{lo}(2) + 2^{29}a^{hi}(3),\\
\texttt{rol}_5(W)^{lo} + 2^{16}\texttt{rol}_5(W)^{hi} &= a^{lo}(2) + 2^{2}a^{hi}(3) + 2^{5}c(16) + 2^{21}b(11),\\
\texttt{range\_check}(a^{lo}(2), 0, 3) &= 0,\\
\texttt{range\_check}(a^{hi}(3), 0, 7) &= 0,\\
\texttt{range\_check}(\text{tag}(b(11), 0, 3) &= 0.
\end{aligned}$$

The last three constraints have the following explanation.

- Restricting $a^{lo}(2)$ to the range 0 to 3 forces $a^{lo}(2)$ to be a 2-bit value.
- Restricting $a^{hi}(3)$ to the range 0 to 7 forces $a^{hi}(3)$ to be a 3-bit value.
- Restricting the tag of $b(11)$ to the range 0 to 3 forces $b(11)$ to be a 11-bit value.

3.2.2 $\texttt{rol}_6$ gate

We decompose $W = w_{31}w_{30}\cdots w_2w_1w_0$ into four parts $a^{hi}(3), a^{lo}(3), b(10), c(16)$,

$$\underbrace{w_{31}\cdots w_{29}}_{a^{hi}(3)}\underbrace{w_{28}\cdots w_{26}}_{a^{lo}(3)}\underbrace{w_{25}w_{24}w_{23}\cdots w_{16}}_{b(10)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)},$$

where the numbers in parentheses indicate lengths of the respective parts in bits. The rotated word in terms of these four parts is given by

$$\texttt{rol}_6(W) = \underbrace{w_{26}w_{25}w_{24}\cdots w_{16}}_{b(10)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)}\underbrace{w_{31}\cdots w_{29}}_{a^{hi}(3)}\underbrace{w_{28}\cdots w_{26}}_{a^{lo}(3)}.$$

The layout of the gate is as shown below, where `s_rotate_left[1]` is the selector that activates it.

`s_rotate_left[1]`	a_0	a_1	a_2	a_3	a_4	a_5
1	tag($b(10)$)	$b(10)$		$a^{lo}(3)$	W^{lo}	$\texttt{rol}_6(W)^{lo}$
		$c(16)$		$a^{hi}(3)$	W^{hi}	$\texttt{rol}_6(W)^{hi}$

The following constraints are enforced by the gate where $\texttt{range\_check}(x, j, k) = \prod_{i=j}^{k}(x - i)$.

$$\begin{aligned} W^{lo} + 2^{16}W^{hi} &= c(16) + 2^{16}b(10) + 2^{26}a^{lo}(3) + 2^{29}a^{hi}(3), \\ \texttt{rol}_6(W)^{lo} + 2^{16}\texttt{rol}_6(W)^{hi} &= a^{lo}(3) + 2^{3}a^{hi}(3) + 2^{6}c(16) + 2^{22}b(10), \\ \texttt{range\_check}(a^{lo}(3), 0, 7) &= 0, \\ \texttt{range\_check}(a^{hi}(3), 0, 7) &= 0, \\ \texttt{range\_check}(\text{tag}(b(10), 0, 2) &= 0. \end{aligned}$$

The last three constraints have the following explanation.

- Restricting $a^{lo}(3)$ to the range 0 to 3 forces $a^{lo}(3)$ to be a 3-bit value.
- Restricting $a^{hi}(3)$ to the range 0 to 7 forces $a^{hi}(3)$ to be a 3-bit value.
- Restricting the tag of $b(10)$ to the range 0 to 2 forces $b(10)$ to be a 10-bit value.

3.2.3 $\texttt{rol}_7$ gate

We decompose $W = w_{31}w_{30}\cdots w_2w_1w_0$ into four parts $a^{hi}(4), a^{lo}(3), b(9), c(16)$,

$$\underbrace{w_{31}\cdots w_{28}}_{a^{hi}(4)}\underbrace{w_{27}\cdots w_{25}}_{a^{lo}(3)}\underbrace{w_{24}w_{23}w_{22}\cdots w_{16}}_{b(9)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)},$$

where the numbers in parentheses indicate lengths of the respective parts in bits. The rotated word in terms of these four parts is given by

$$\texttt{rol}_7(W) = \underbrace{w_{24}w_{23}w_{22}\cdots w_{16}}_{b(9)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)}\underbrace{w_{31}\cdots w_{28}}_{a^{hi}(4)}\underbrace{w_{27}\cdots w_{25}}_{a^{lo}(3)}.$$

The layout of the gate is as shown below, where `s_rotate_left[2]` is the selector that activates it.

`s_rotate_left[2]`	a_0	a_1	a_2	a_3	a_4	a_5
1	tag($b(9)$)	$b(9)$		$a^{lo}(3)$	W^{lo}	$\texttt{rol}_7(W)^{lo}$
		$c(16)$		$a^{hi}(4)$	W^{hi}	$\texttt{rol}_7(W)^{hi}$

The following constraints are enforced by the gate where $\texttt{range\_check}(x, j, k) = \prod_{i=j}^{k}(x - i)$.

$$\begin{aligned} W^{lo} + 2^{16}W^{hi} &= c(16) + 2^{16}b(10) + 2^{25}a^{lo}(3) + 2^{28}a^{hi}(4), \\ \texttt{rol}_7(W)^{lo} + 2^{16}\texttt{rol}_7(W)^{hi} &= a^{lo}(3) + 2^{3}a^{hi}(4) + 2^{7}c(16) + 2^{23}b(10), \\ \texttt{range\_check}(a^{lo}(3), 0, 7) &= 0, \\ \texttt{range\_check}(a^{hi}(4), 0, 15) &= 0, \\ \texttt{range\_check}(\text{tag}(b(9), 0, 1) &= 0. \end{aligned}$$

The last three constraints have the following explanation.

- Restricting $a^{lo}(3)$ to the range 0 to 7 forces $a^{lo}(3)$ to be a 3-bit value.
- Restricting $a^{hi}(4)$ to the range 0 to 15 forces $a^{hi}(4)$ to be a 4-bit value.
- Restricting the tag of $b(9)$ to the range 0 to 1 forces $b(9)$ to be a 9-bit value.

3.2.4 $\texttt{rol}_8$ gate

We decompose $W = w_{31}w_{30}\cdots w_2w_1w_0$ into four parts $a^{hi}(4), a^{lo}(4), b(9), c(16)$,

$$\underbrace{w_{31}\cdots w_{28}}_{a^{hi}(4)}\underbrace{w_{27}\cdots w_{24}}_{a^{lo}(4)}\underbrace{w_{23}w_{22}w_{21}\cdots w_{16}}_{b(8)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)},$$

where the numbers in parentheses indicate lengths of the respective parts in bits. The rotated word in terms of these four parts is given by

$$\texttt{rol}_8(W) = \underbrace{w_{23}w_{22}w_{21}\cdots w_{16}}_{b(8)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)}\underbrace{w_{31}\cdots w_{28}}_{a^{hi}(4)}\underbrace{w_{27}\cdots w_{24}}_{a^{lo}(4)}.$$

The layout of the gate is as shown below, where `s_rotate_left[3]` is the selector that activates it.

`s_rotate_left[3]`	a_0	a_1	a_2	a_3	a_4	a_5
1	tag($b(8)$)	$b(8)$		$a^{lo}(4)$	W^{lo}	$\texttt{rol}_8(W)^{lo}$
		$c(16)$		$a^{hi}(4)$	W^{hi}	$\texttt{rol}_8(W)^{hi}$

The following constraints are enforced by the gate where $\texttt{range\_check}(x, j, k) = \prod_{i=j}^{k}(x - i)$.

$$\begin{aligned}
W^{lo} + 2^{16}W^{hi} &= c(16) + 2^{16}b(10) + 2^{24}a^{lo}(4) + 2^{28}a^{hi}(4),\\
\texttt{rol}_8(W)^{lo} + 2^{16}\texttt{rol}_8(W)^{hi} &= a^{lo}(4) + 2^4a^{hi}(4) + 2^8c(16) + 2^{24}b(8),\\
\texttt{range\_check}(a^{lo}(4), 0, 15) &= 0,\\
\texttt{range\_check}(a^{hi}(4), 0, 15) &= 0,\\
\texttt{range\_check}(\text{tag}(b(8), 0, 0) &= 0.
\end{aligned}$$

The last three constraints have the following explanation.

- Restricting $a^{lo}(4)$ to the range 0 to 15 forces $a^{lo}(4)$ to be a 4-bit value.
- Restricting $a^{hi}(4)$ to the range 0 to 15 forces $a^{hi}(4)$ to be a 4-bit value.
- Restricting the tag of $b(8)$ to the range 0 to 1 forces $b(8)$ to be a 8-bit value.

3.2.5 $\texttt{rol}_9$ gate

We decompose $W = w_{31}w_{30}\cdots w_2w_1w_0$ into four parts $a(9), b^{hi}(4), b^{lo}(3), c(16)$,

$$\underbrace{w_{31}w_{30}w_{29}\cdots w_{23}}_{a(9)}\underbrace{w_{22}\cdots w_{19}}_{b^{hi}(4)}\underbrace{w_{18}\cdots w_{16}}_{b^{lo}(3)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)},$$

where the numbers in parentheses indicate lengths of the respective parts in bits. The rotated word in terms of these four parts is given by

$$\texttt{rol}_9(W) = \underbrace{w_{22}\cdots w_{19}}_{b^{hi}(4)}\underbrace{w_{18}\cdots w_{16}}_{b^{lo}(3)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)}\underbrace{w_{31}w_{30}w_{29}\cdots w_{23}}_{a(9)}.$$

The layout of the gate is as shown below, where `s_rotate_left[4]` is the selector that activates it.

`s_rotate_left[4]`	a_0	a_1	a_2	a_3	a_4	a_5
1	tag($a(9)$)	$a(9)$		$b^{lo}(3)$	W^{lo}	$\texttt{rol}_9(W)^{lo}$
		$c(16)$		$b^{hi}(4)$	W^{hi}	$\texttt{rol}_9(W)^{hi}$

The following constraints are enforced by the gate where $\texttt{range\_check}(x, j, k) = \prod_{i=j}^{k}(x-i)$.

$$\begin{aligned}
W^{lo} + 2^{16}W^{hi} &= c(16) + 2^{16}b^{lo}(3) + 2^{19}b^{hi}(4) + 2^{23}a(9),\\
\texttt{rol}_9(W)^{lo} + 2^{16}\texttt{rol}_9(W)^{hi} &= a(9) + 2^{9}c(16) + 2^{25}b^{lo}(3) + 2^{28}b^{hi}(4),\\
\texttt{range\_check}(b^{lo}(3), 0, 7) &= 0,\\
\texttt{range\_check}(b^{hi}(4), 0, 15) &= 0,\\
\texttt{range\_check}(\text{tag}(a(9), 0, 1) &= 0.
\end{aligned}$$

The last three constraints have the following explanation.

- Restricting $b^{lo}(3)$ to the range 0 to 7 forces $b^{lo}(3)$ to be a 3-bit value.
- Restricting $b^{hi}(4)$ to the range 0 to 15 forces $b^{hi}(4)$ to be a 4-bit value.
- Restricting the tag of $a(9)$ to the range 0 to 1 forces $a(9)$ to be a 9-bit value.

3.2.6 $\texttt{rol}_{10}$ gate

We decompose $W = w_{31}w_{30}\cdots w_2w_1w_0$ into four parts $a(10), b^{hi}(3), b^{lo}(3), c(16)$,

$$\underbrace{w_{31}w_{30}w_{29}\cdots w_{22}}_{a(10)}\underbrace{w_{21}\cdots w_{19}}_{b^{hi}(3)}\underbrace{w_{18}\cdots w_{16}}_{b^{lo}(3)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)},$$

where the numbers in parentheses indicate lengths of the respective parts in bits. The rotated word in terms of these four parts is given by

$$\texttt{rol}_{10}(W) = \underbrace{w_{21}\cdots w_{19}}_{b^{hi}(3)}\underbrace{w_{18}\cdots w_{16}}_{b^{lo}(3)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)}\underbrace{w_{31}w_{30}w_{29}\cdots w_{22}}_{a(10)}.$$

The layout of the gate is as shown below, where `s_rotate_left[5]` is the selector that activates it.

`s_rotate_left[5]`	a_0	a_1	a_2	a_3	a_4	a_5
1	tag($a(10)$)	$a(10)$		$b^{lo}(3)$	W^{lo}	$\texttt{rol}_{10}(W)^{lo}$
		$c(16)$		$b^{hi}(3)$	W^{hi}	$\texttt{rol}_{10}(W)^{hi}$

The following constraints are enforced by the gate where $\texttt{range\_check}(x, j, k) = \prod_{i=j}^{k}(x - i)$.

$$\begin{aligned}
W^{lo} + 2^{16}W^{hi} &= c(16) + 2^{16}b^{lo}(3) + 2^{19}b^{hi}(3) + 2^{22}a(10),\\
\texttt{rol}_{10}(W)^{lo} + 2^{16}\texttt{rol}_{10}(W)^{hi} &= a(10) + 2^{10}c(16) + 2^{26}b^{lo}(3) + 2^{29}b^{hi}(3),\\
\texttt{range\_check}(b^{lo}(3), 0, 7) &= 0,\\
\texttt{range\_check}(b^{hi}(3), 0, 7) &= 0,\\
\texttt{range\_check}(\text{tag}(a(10), 0, 2) &= 0.
\end{aligned}$$

The last three constraints have the following explanation.

- Restricting $b^{lo}(3)$ to the range 0 to 7 forces $b^{lo}(3)$ to be a 3-bit value.
- Restricting $b^{hi}(3)$ to the range 0 to 7 forces $b^{hi}(3)$ to be a 3-bit value.
- Restricting the tag of $a(10)$ to the range 0 to 2 forces $a(10)$ to be a 10-bit value.

3.2.7 $\texttt{rol}_{11}$ gate

We decompose $W = w_{31}w_{30}\cdots w_2w_1w_0$ into four parts $a(11), b^{hi}(3), b^{lo}(2), c(16)$,

$$\underbrace{w_{31}w_{30}w_{29}\cdots w_{21}}_{a(11)}\underbrace{w_{20}\cdots w_{18}}_{b^{hi}(3)}\underbrace{w_{17}w_{16}}_{b^{lo}(2)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)},$$

where the numbers in parentheses indicate lengths of the respective parts in bits. The rotated word in terms of these four parts is given by

$$\texttt{rol}_{11}(W) = \underbrace{w_{20}\cdots w_{18}}_{b^{hi}(3)}\underbrace{w_{17}w_{16}}_{b^{lo}(2)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)}\underbrace{w_{31}w_{30}w_{29}\cdots w_{21}}_{a(11)}.$$

The layout of the gate is as shown below, where `s_rotate_left[6]` is the selector that activates it.

`s_rotate_left[6]`	a_0	a_1	a_2	a_3	a_4	a_5
1	tag($a(11)$)	$a(11)$		$b^{lo}(2)$	W^{lo}	$\texttt{rol}_{11}(W)^{lo}$
		$c(16)$		$b^{hi}(3)$	W^{hi}	$\texttt{rol}_{11}(W)^{hi}$

The following constraints are enforced by the gate where $\texttt{range\_check}(x, j, k) = \prod_{i=j}^{k}(x - i)$.

$$\begin{aligned}
W^{lo} + 2^{16}W^{hi} &= c(16) + 2^{16}b^{lo}(2) + 2^{18}b^{hi}(3) + 2^{21}a(11),\\
\texttt{rol}_{11}(W)^{lo} + 2^{16}\texttt{rol}_{11}(W)^{hi} &= a(11) + 2^{11}c(16) + 2^{27}b^{lo}(2) + 2^{29}b^{hi}(3),\\
\texttt{range\_check}(b^{lo}(2), 0, 3) &= 0,\\
\texttt{range\_check}(b^{hi}(3), 0, 7) &= 0,\\
\texttt{range\_check}(\text{tag}(a(11), 0, 3) &= 0.
\end{aligned}$$

The last three constraints have the following explanation.

- Restricting $b^{lo}(2)$ to the range 0 to 3 forces $b^{lo}(2)$ to be a 2-bit value.
- Restricting $b^{hi}(3)$ to the range 0 to 7 forces $b^{hi}(3)$ to be a 3-bit value.
- Restricting the tag of $a(11)$ to the range 0 to 3 forces $a(11)$ to be a 11-bit value.

3.2.8 $\texttt{rol}_{12}$ gate

We decompose $W = w_{31}w_{30}\cdots w_2w_1w_0$ into four parts $a(12), b^{hi}(2), b^{lo}(2), c(16)$,

$$\underbrace{w_{31}w_{30}w_{29}\cdots w_{20}}_{a(12)}\underbrace{w_{19}w_{18}}_{b^{hi}(2)}\underbrace{w_{17}w_{16}}_{b^{lo}(2)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)},$$

where the numbers in parentheses indicate lengths of the respective parts in bits. The rotated word in terms of these four parts is given by

$$\texttt{rol}_{12}(W) = \underbrace{w_{19}w_{18}}_{b^{hi}(2)}\underbrace{w_{17}w_{16}}_{b^{lo}(2)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)}\underbrace{w_{31}w_{30}w_{29}\cdots w_{20}}_{a(12)}.$$

The layout of the gate is as shown below, where `s_rotate_left[7]` is the selector that activates it.

`s_rotate_left[7]`	a_0	a_1	a_2	a_3	a_4	a_5
1	tag($a(12)$)	$a(12)$		$b^{lo}(2)$	W^{lo}	$\texttt{rol}_{12}(W)^{lo}$
		$c(16)$		$b^{hi}(3)$	W^{hi}	$\texttt{rol}_{12}(W)^{hi}$

The following constraints are enforced by the gate where $\texttt{range\_check}(x, j, k) = \prod_{i=j}^{k}(x - i)$.

$$\begin{aligned}
W^{lo} + 2^{16}W^{hi} &= c(16) + 2^{16}b^{lo}(2) + 2^{18}b^{hi}(2) + 2^{20}a(12),\\
\texttt{rol}_{12}(W)^{lo} + 2^{16}\texttt{rol}_{12}(W)^{hi} &= a(12) + 2^{12}c(16) + 2^{28}b^{lo}(2) + 2^{30}b^{hi}(2),\\
\texttt{range\_check}(b^{lo}(2), 0, 3) &= 0,\\
\texttt{range\_check}(b^{hi}(2), 0, 3) &= 0,\\
\texttt{range\_check}(\text{tag}(a(12), 0, 4) &= 0.
\end{aligned}$$

The last three constraints have the following explanation.

- Restricting $b^{lo}(2)$ to the range 0 to 3 forces $b^{lo}(2)$ to be a 2-bit value.
- Restricting $b^{hi}(2)$ to the range 0 to 3 forces $b^{hi}(2)$ to be a 2-bit value.
- Restricting the tag of $a(12)$ to the range 0 to 4 forces $a(12)$ to be a 12-bit value.

3.2.9 $\texttt{rol}_{13}$ gate

We decompose $W = w_{31}w_{30}\cdots w_2w_1w_0$ into three parts $a(13), b(3), c(16)$,

$$\underbrace{w_{31}w_{30}w_{29}\cdots w_{19}}_{a(13)}\underbrace{w_{18}\cdots w_{16}}_{b(3)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)},$$

where the numbers in parentheses indicate lengths of the respective parts in bits. The rotated word in terms of these four parts is given by

$$\texttt{rol}_{13}(W) = \underbrace{w_{18}\cdots w_{16}}_{b(3)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)}\underbrace{w_{31}w_{30}w_{29}\cdots w_{19}}_{a(13)}.$$

The layout of the gate is as shown below, where `s_rotate_left[8]` is the selector that activates it.

s_rotate_left[8]	a_0	a_1	a_2	a_3	a_4	a_5
1	tag($a(13)$)	$a(13)$		$b(3)$	W^{lo}	$\texttt{rol}_{13}(W)^{lo}$
		$c(16)$			W^{hi}	$\texttt{rol}_{13}(W)^{hi}$

The following constraints are enforced by the gate where $\texttt{range\_check}(x, j, k) = \prod_{i=j}^{k}(x - i)$.

$$\begin{aligned}
W^{lo} + 2^{16}W^{hi} &= c(16) + 2^{16}b(3) + 2^{19}a(13), \\
\texttt{rol}_{13}(W)^{lo} + 2^{16}\texttt{rol}_{13}(W)^{hi} &= a(13) + 2^{13}c(16) + 2^{29}b(3), \\
\texttt{range\_check}(b(3), 0, 7) &= 0, \\
\texttt{range\_check}(\text{tag}(a(13), 0, 5) &= 0.
\end{aligned}$$

The last two constraints have the following explanation.

- Restricting $b(3)$ to the range 0 to 7 forces $b(3)$ to be a 3-bit value.
- Restricting the tag of $a(13)$ to the range 0 to 5 forces $a(13)$ to be a 13-bit value.

3.2.10 $\texttt{rol}_{14}$ gate

We decompose $W = w_{31}w_{30}\cdots w_2w_1w_0$ into three parts $a(14), b(2), c(16)$,

$$\underbrace{w_{31}w_{30}w_{29}\cdots w_{18}}_{a(14)}\underbrace{w_{17}w_{16}}_{b(2)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)},$$

where the numbers in parentheses indicate lengths of the respective parts in bits. The rotated word in terms of these four parts is given by

$$\texttt{rol}_{13}(W) = \underbrace{w_{17}w_{16}}_{b(2)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)}\underbrace{w_{31}w_{30}w_{29}\cdots w_{18}}_{a(14)}.$$

The layout of the gate is as shown below, where s_rotate_left[9] is the selector that activates it.

s_rotate_left[9]	a_0	a_1	a_2	a_3	a_4	a_5
1	tag($a(14)$)	$a(14)$		$b(2)$	W^{lo}	$\texttt{rol}_{14}(W)^{lo}$
		$c(16)$			W^{hi}	$\texttt{rol}_{14}(W)^{hi}$

The following constraints are enforced by the gate where $\texttt{range\_check}(x, j, k) = \prod_{i=j}^{k}(x - i)$.

$$\begin{aligned}
W^{lo} + 2^{16}W^{hi} &= c(16) + 2^{16}b(2) + 2^{18}a(14), \\
\texttt{rol}_{14}(W)^{lo} + 2^{16}\texttt{rol}_{14}(W)^{hi} &= a(14) + 2^{14}c(16) + 2^{30}b(2), \\
\texttt{range\_check}(b(2), 0, 3) &= 0, \\
\texttt{range\_check}(\text{tag}(a(14), 0, 6) &= 0.
\end{aligned}$$

The last two constraints have the following explanation.

- Restricting $b(2)$ to the range 0 to 3 forces $b(2)$ to be a 2-bit value.
- Restricting the tag of $a(14)$ to the range 0 to 6 forces $a(14)$ to be a 14-bit value.

3.2.11 `rol`$_{15}$ gate

We decompose $W = w_{31}w_{30}\cdots w_2w_1w_0$ into three parts $a(15), b(1), c(16)$,

$$\underbrace{w_{31}w_{30}w_{29}\cdots w_{18}}_{a(15)}\underbrace{w_{16}}_{b(1)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)},$$

where the numbers in parentheses indicate lengths of the respective parts in bits. The rotated word in terms of these four parts is given by

$$\texttt{rol}_{13}(W) = \underbrace{w_{16}}_{b(1)}\underbrace{w_{15}w_{14}w_{13}w_{12}\cdots w_0}_{c(16)}\underbrace{w_{31}w_{30}w_{29}\cdots w_{18}}_{a(14)}.$$

The layout of the gate is as shown below, where `s_rotate_left[10]` is the selector that activates it.

`s_rotate_left[10]`	a_0	a_1	a_2	a_3	a_4	a_5
1	$\text{tag}(a(15))$	$a(15)$		$b(1)$	W^{lo}	$\texttt{rol}_{15}(W)^{lo}$
		$c(16)$			W^{hi}	$\texttt{rol}_{15}(W)^{hi}$

The following constraints are enforced by the gate where $\texttt{range\_check}(x, j, k) = \prod_{i=j}^{k}(x - i)$.

$$\begin{aligned}
W^{lo} + 2^{16}W^{hi} &= c(16) + 2^{16}b(1) + 2^{17}a(15),\\
\texttt{rol}_{15}(W)^{lo} + 2^{16}\texttt{rol}_{15}(W)^{hi} &= a(15) + 2^{15}c(16) + 2^{31}b(1),\\
\texttt{range\_check}(b(1), 0, 1) &= 0,\\
\texttt{range\_check}(\text{tag}(a(15), 0, 7) &= 0.
\end{aligned}$$

The last two constraints have the following explanation.

- Restricting $b(1)$ to the range 0 to 1 forces $b(1)$ to be a 1-bit value.
- Restricting the tag of $a(15)$ to the range 0 to 7 forces $a(15)$ to be a 15-bit value.

References

[1] Ronald L. Rivest. The MD4 message digest algorithm. In *Advances in Cryptology-CRYPTO' 90*, pages 303–311, 1991.