Mempuf Project – Tarek Idriss

Marcaus Cruz

July 2021

1 Types of PUFs

1.1 Mempuf

The Memory PUF (Mempuf for short) is the core PUF of this project as it aims to use, to it's advantage, a component of hardware that is present in virtually all devices- **memory**. The architecture of the Mempuf is standard; having fields like dimension (bit size of puf), data size (number of CRPs), challenge and response arrays, a weights array for uniqueness, and a couple of added fields in a phi array (for Machine Learning purposes) and a 'weight bits' value (used to control the range of weight values). One last field that makes this type of PUF stand out from the others is the T array, which is used in this particular evaluation process.

Enter Equation here

The Mempuf follows this equation for it's evaluation which constantly uses the XOR of previous challenge bits (T value) to sum up to an overall response once it reaches n, the dimension of the PUF. This sum will ultimately result in a positive or negative number which corresponds to a 1 or 0 as the response bit, respectively.

1.2 Feed-Forward Mempuf

The Feed-Forward Mempuf (FF Mempuf) behaves very similarly to the Mempuf. They have the same fields to them, but differ slightly during the evaluation process.

Enter Equation here

As shown above in the above equation, this PUF adds one more XOR element to the T value during evaluation at index i, which is the sign of the sum at index *i-1*. This sign, either positive or negative, corresponds to XORing the T value with a 1 or 0. This process is known as *feeding forward* the sign of the previous sum to add an extra element of intricacy during evaluation.

1.3 XOR Mempuf

The XOR Mempuf adopts the same fields as the Mempuf as well as an array of a specified number of Mempufs indicated by a *–rows* flag. Something to note is that the challenges of these individual Mempufs are uniform. The evaluation of the XOR Mempuf is a unique one as it first requires all of it's internal Mempufs to have CRPs generated before it can start it's own evaluation. The Puf then XORs the **0...nth** bit of all responses from all individual Mempufs to generate what we call an *Overall Response*. The XOR Mempuf's 'responses' array is then set to the batch of overall responses calculated in the evaluation.

1.3.1 XOR Feed-Forward Mempuf

As it's name suggests, this PUF is almost identical to the XOR Mempuf. Instead of Mempufs as the internal PUFs, we use the previously mentioned FF Mempufs for heightened complexity.

1.4 Control Feed-Forward Mempuf

The Control Feed-Forward Mempuf (Ctrl FF Mempuf) is a unification of the Mempuf and the FF Mempuf. Per usual, the fields of this PUF remain the same as a Mempuf except for the addition of a *loop* string, a *loop_start* array, and a *loop_end* array. The loop is a string of paired numbers indicating the desired bit in the challenge to be fed forward to the desired index during evaluation.

Here's an example of a valid loop string:

"1 52 19 25 15 24 60 64 37 43"

As shown, the first of the paired indices must be less than it's counterpart because you cannot feed backwards during the evaluation. This string is telling the Ctrl FF Mempuf that during evaluation it should feed the sign of the sum at index 1 to XOR with the T value at index 52, the sign of the sum at index 19 to XOR with the T value at index 25, and so on and so forth. There's a check in the evaluation of a challenge that if the current index that we're at in the challenge is in the loop_end array, we feed the corresponding saved value forward. Otherwise, this PUF is evaluated as a normal Mempuf.

2 Special Flags Developed

2.1 –weight_bits: int

Weights are what make these soft replicas of a PUF unique when evaluating challenges. With a default value of 4 bits, the weights range from -8 to 7. To add variability to weights to potentially increase complexity, thus security, we added this flag which will increase or decrease the range depending on if you set the weight bits greater than or less than 4, respectively.

2.2 –rows: int

This flag is only to be used when working with XOR Mempufs or XOR FF Mempufs. It dictates the number of internal PUFs generated within the XOR PUF. There is no default value for this flag, so it's important to ensure the usage of this flag when dealing with XOR PUFs, otherwise the PUF would act as a normal Mempuf or FF Mempuf. For benchmarking, we used 3 rows and 5 rows.

2.3 –ctrl_loop: string

The loop string is used only for the Ctrl FF Mempuf which indicates the indices you want to feed from and to during evaluation. This flag was added to increase intricacy of the Mempuf as ultimately succeeded. There is no default value for this flag, so using this flag is necessary when handling the Ctrl FF Mempuf.

2.4 Flip bit flags

These flags are used when testing reliability through bit flipping. They're function right now is to flip a single bit per run, but could be used to flip multiple bits in different weights. Little additional code would be needed to do this.

2.4.1 –flip_weight: int

Indicates which weight index to manipulate a bit of. The range is [0 ... dimension - 1].
Special flags needed to pair with:

- –flip_bit

2.4.2 –flip_bit: int

Specifies the bit index of the specified weight to flip. The range is [0 ... weight_bits - 1].
Special flags needed to pair with:

- –flip_weight

2.4.3 –xor_flip: int

Specifies the sub-PUF of any xor PUF to do bit manipulation on. The range is [0 ... rows - 1].
Special flags needed to pair with:

- –flip_bit
- –flip_weight
- –rows

If you wanted to flip the first bit of the fifteenth weight in the third sub-PUF: –flip_weight 14 –flip_bit 0 –xor_flip 2