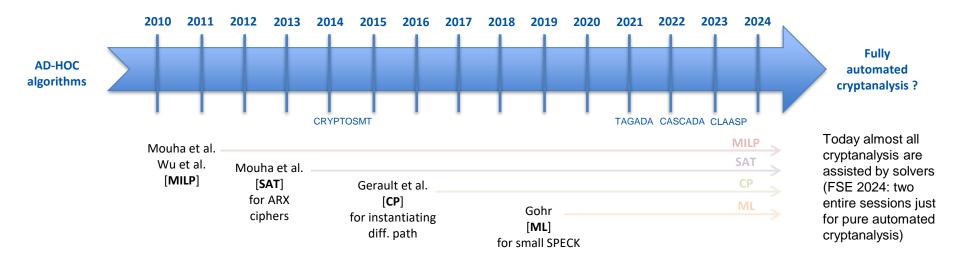


Open Cryptanalysis Platform (OCP) A New Collaborative Effort for Automated Cryptanalysis

Thomas Peyrin – Chunning Zhou NTU Singapore

SKCAM 2025 - Roma 15th March 2025

Timeline of Automated Cryptanalysis



Automated cryptanalysis using declarative frameworks (SAT/MILP/CP/etc.) is generally slower or at best same as ad-hoc tools, but so much more convenient

Mainly on **differential** and **linear cryptanalysis**, but now also on integral distinguishers, cube attacks, meet-in-the-middle attacks, etc.

Solving time is a crucial aspect and can be impacted by:

- the framework you use (SAT/MILP/CP/etc.)
- the strategy of modeling (many works on various modeling strategies)
- the solver (less contributions on that, different research field)
- the type of problem studied / scale



Open Cryptanalysis Platform (Open-CP)



NANYANG TECHNOLOGICAL UNIVERSITY | SINGAPORE

Open-CP: a new collaborative cryptanalysis platform

- Automatic generation of attacks / implementations
- In collaboration with all cryptanalysts willing to join !
- Free and open source
- Easy to <u>use</u> and <u>contribute</u> (define a common language we all understand)
- Very modular, allows custom operators, etc.
- Easy to install (no or very minimal use of external tools)
- Start simple (differential)
- <u>Goal</u>: become the go-to platform for creating / testing / benchmarking cryptanalysis

https://github.com/Open-CP/OCP



How to Define a Crypto Scheme ?

First step: define a common language we all agree with and understand. Once we have that, a LOT of things are possible.

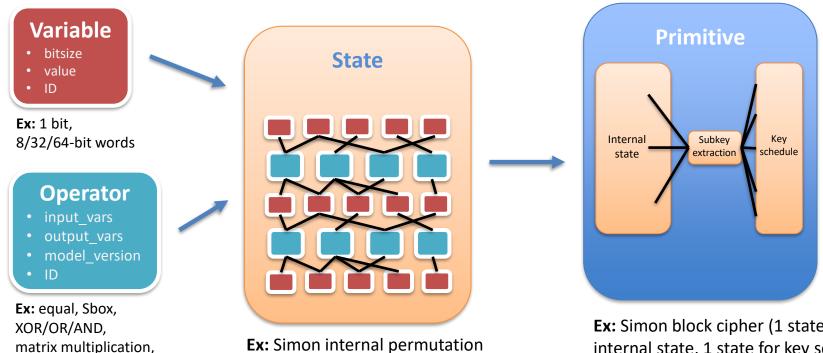
How do we **describe** a cipher in a generic way and have this language usable for our global cryptanalysis platform ?



Current Architecture of Open-CP

- Coded in Python: everything is an object and can have its own modeling
- Internal representation: a multipartite directed acyclic graph

 similar to TAGADA and CLAASP



Ex: Simon internal permutation Simon key schedule

Ex: Simon block cipher (1 state for internal state, 1 state for key schedule state, 1 state for subkey extraction)



modular addition, ...

State Architecture of Open-CP

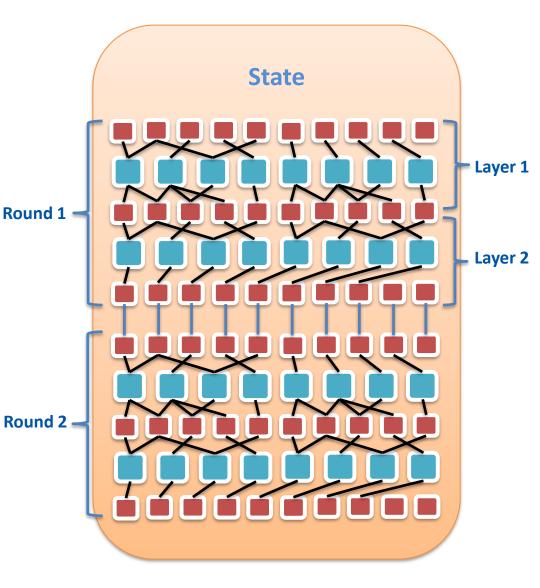
A **state** is represented by a **collection of rounds**, each round is composed of a **collection of layers.**

Advantage: this allows very easy indexing and usage

vars[r][l][i] for variable at
round r, layer l, position i

Drawback: some redundant variables, but not a problem, we can clean after if needed

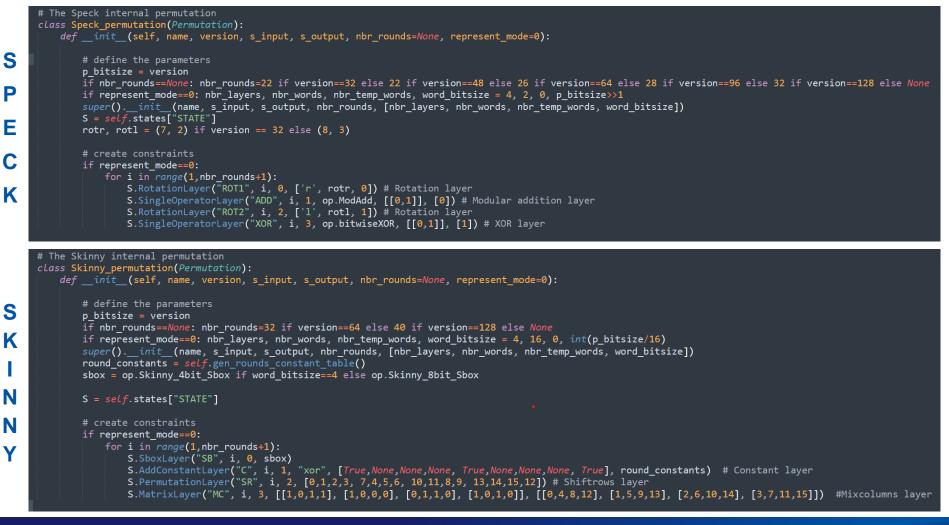
Ex: AES internal state has 10 rounds of 4 layers each (SB, SR, MC/ID, AC) with 16 variables of 8 bits each.





Easy and Fast cipher definition

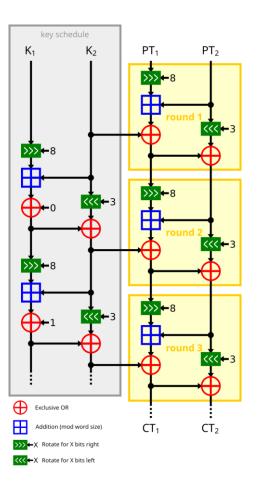
Many functions already present to help you define a state very easily



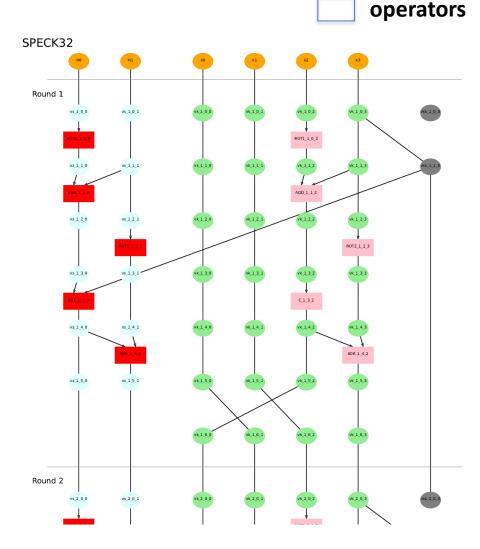


NANYANG TECHNOLOGICAL UNIVERSITY | SINGAPORE

Modeling Example of Open-CP



Example: SPECK-32 block cipher

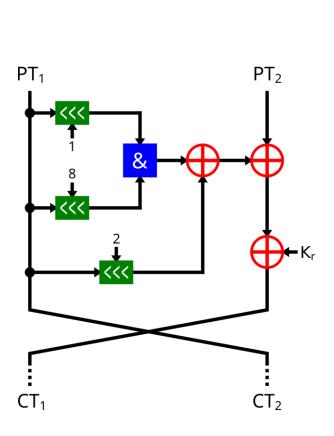


(image from Wikipedia)



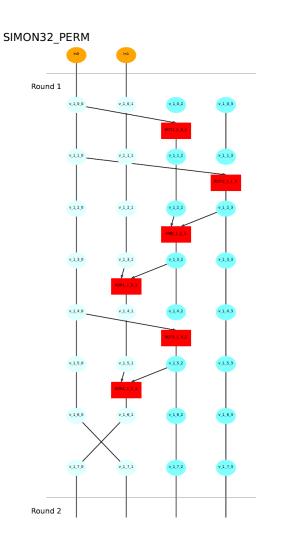
variables

Modeling Example of Open-CP



Example: SIMON-32 permutation

(image from Wikipedia)

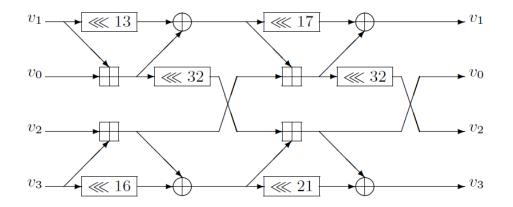


variables

operators



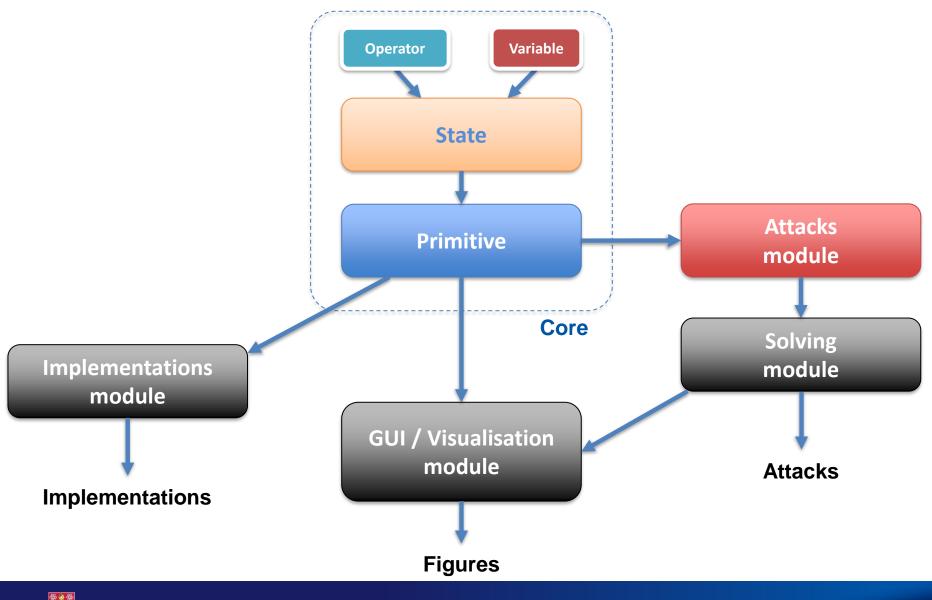
Ex: SipHash



```
# The SipHash internal permutation
class SipHash_permutation(Permutation):
   def init (self, name, s input, s output, nbr rounds=None, represent mode=0):
       nbr layers = 10
       nbr words = 4
       nbr temp words = 0
       word bitsize = 64
       super(). init (name, s_input, s_output, nbr_rounds, [nbr_layers, nbr_words, nbr_temp_words, word_bitsize])
       S = self.states["STATE"]
       # create constraints
       if represent mode==0:
           for i in range(1,nbr_rounds+1):
               S.SingleOperatorLayer("ADD1", i, 0, op.ModAdd, [[0,1], [2,3]], [0, 2]) # Modular addition layer
               S.RotationLayer("ROT1", i, 1, [['1', 13, 1], ['1', 16, 3]]) # Rotation layer
               S.SingleOperatorLayer("XOR1", i, 2, op.bitwiseXOR, [[0,1], [2,3]], [1, 3]) # XOR layer
               S.RotationLayer("ROT2", i, 3, [['1', 32, 0]]) # Rotation layer
               S.PermutationLayer("PERM1", i, 4, [2,1,0,3]) # Permutation layer
               S.SingleOperatorLayer("ADD2", i, 5, op.ModAdd, [[0,1], [2,3]], [0, 2]) # Modular addition layer
               S.RotationLayer("ROT3", i, 6, [['1', 17, 1], ['1', 21, 3]]) # Rotation layer
               S.SingleOperatorLayer("XOR2", i, 7, op.bitwiseXOR, [[0,1], [2,3]], [1, 3]) # XOR layer
               S.RotationLayer("ROT4", i, 8, [['1', 32, 0]]) # Rotation layer
               S.PermutationLayer("PERM2", i, 9, [2,1,0,3]) # Permutation layer
```



Architecture of Open-CP Tool



NANYANG TECHNOLOGICAL UNIVERSITY | SINGAPORE

Automatic Generation of C / Python code

itsize itsize

<pre>#Notation Macros def ROT(0, d, bitsize): return ((n << d) (n >> (bitsize - d))) & (2*bitsize): return ((n >> d) (n << (bitsize - d))) & (2*bitsize): return ((n >> d) (n << (bitsize - d))) & (2*bitsize): return ((n >> d) (n << (bitsize - d))) & (2*bitsize): return ((n >> d) (n << (bitsize - d))) & (2*bitsize): return ((n >> d) (n << (bitsize - d))) & (2*bitsize): return ((n >> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize - d))) & (2*bitsize): return ((n <> d) (n << (bitsize): return ((n <> n) (n << (bitsize): return ((n << n) (n << n) (n <<</pre>	
<pre>4 # Function implementing the SIMON32_PERM function # Input: # Input: # Output: # Output: # Output: # Output: # Input * 0.0 = IN[0] * 0.0 = IN[0] * 0.0 = IN[0] * 0.0 = 0.0 * 1.0 = v.0.0 * 1.0 = v.0.0 * 1.1 = v.0.1 * Round function # Round function # Round function * v.1.2 = ROI1(v.0.0, 1, 16) * 1.2 = ROI1(v.0.0, 1, 16) * v.1.2 = ROI1(v.0.0, 1, 16) * v.1.3 = v.0.3 * v.2.3 = ROI1(v.1.0, 8, 16) * v.3.0 = v.2.8 * v.3.1 = v.1.1 * v.2.2 = ROI1(v.1.0, 8, 16) * v.3.0 = v.2.8 * v.3.1 = v.2.1 * v.3.2 = v.2.2 & v.2.3 * v.3.3 = v.2.3 * v.4.0 = v.3.0 * v.4.1 = v.3.1 * v.4.2 = v.3.2 * v.4.2 = v.3.2 * v.4.2 = v.3.2 * v.4.2 = v.3.3 * v.5.0 = v.4.0 * v.5.1 = v.4.1 * v.5.2 = ROI1(v.4.0, 2, 16) * v.5.2 = ROI1(v.4.0, 2, 16) * v.5.2 = v.4.3 * v.6.0 = v.5.0 * v.6.1 = v.5.1 * v.5.2 * v.6.2 = v.5.3 * v.6.0 = v.5.0 * v.6.1 = v.5.1 * v.5.2 * v.6.2 = v.7.3 * v.6.2 = v.7.3 * v.6.2 = v.7.3 * v.6.2 = v.7.3 * v.6.3 = v.7.3 * v.6.4 = v.7.1 * v.6.2 = v.7.2 * v.6.3 = v.7.3 * v.6.0 = v.7.0 * v.0.1 = v.7.1 * v.0.2 = v.7.3 # Output OUT[0] = v.7.0 * v.0.3 = v.7.3 # Coutput OUT[0] = v.7.0 * v.0.3 = v.7.3 * Output OUT[1] = v.7.1 * test implementation TN = [6x0, 0x0] SIMON32_PERM(IN, OUT) print('OUT', str([hex(i) for i in OUT])) </pre>	
<pre>4 # Function implementing the SIMON32_PERM function # Input: # Input: # Output: # Output: # Output: # Output: # Input * 0.0 = IN[0] * 0.0 = IN[0] * 0.0 = IN[0] * 0.0 = 0.0 * 1.0 = v.0.0 * 1.0 = v.0.0 * 1.1 = v.0.1 * Round function # Round function # Round function * v.1.2 = ROI1(v.0.0, 1, 16) * 1.2 = ROI1(v.0.0, 1, 16) * v.1.2 = ROI1(v.0.0, 1, 16) * v.1.3 = v.0.3 * v.2.3 = ROI1(v.1.0, 8, 16) * v.3.0 = v.2.8 * v.3.1 = v.1.1 * v.2.2 = ROI1(v.1.0, 8, 16) * v.3.0 = v.2.8 * v.3.1 = v.2.1 * v.3.2 = v.2.2 & v.2.3 * v.3.3 = v.2.3 * v.4.0 = v.3.0 * v.4.1 = v.3.1 * v.4.2 = v.3.2 * v.4.2 = v.3.2 * v.4.2 = v.3.2 * v.4.2 = v.3.3 * v.5.0 = v.4.0 * v.5.1 = v.4.1 * v.5.2 = ROI1(v.4.0, 2, 16) * v.5.2 = ROI1(v.4.0, 2, 16) * v.5.2 = v.4.3 * v.6.0 = v.5.0 * v.6.1 = v.5.1 * v.5.2 * v.6.2 = v.5.3 * v.6.0 = v.5.0 * v.6.1 = v.5.1 * v.5.2 * v.6.2 = v.7.3 * v.6.2 = v.7.3 * v.6.2 = v.7.3 * v.6.2 = v.7.3 * v.6.3 = v.7.3 * v.6.4 = v.7.1 * v.6.2 = v.7.2 * v.6.3 = v.7.3 * v.6.0 = v.7.0 * v.0.1 = v.7.1 * v.0.2 = v.7.3 # Output OUT[0] = v.7.0 * v.0.3 = v.7.3 # Coutput OUT[0] = v.7.0 * v.0.3 = v.7.3 * Output OUT[1] = v.7.1 * test implementation TN = [6x0, 0x0] SIMON32_PERM(IN, OUT) print('OUT', str([hex(i) for i in OUT])) </pre>	
<pre>4 # Function implementing the SIMON32_PERM function # Input: # Input: # Output: # Output: # Output: # Output: # Input * 0.0 = IN[0] * 0.0 = IN[0] * 0.0 = IN[0] * 0.0 = 0.0 * 1.0 = v.0.0 * 1.0 = v.0.0 * 1.1 = v.0.1 * Round function # Round function # Round function * v.1.2 = ROI1(v.0.0, 1, 16) * 1.2 = ROI1(v.0.0, 1, 16) * v.1.2 = ROI1(v.0.0, 1, 16) * v.1.3 = v.0.3 * v.2.3 = ROI1(v.1.0, 8, 16) * v.3.0 = v.2.8 * v.3.1 = v.1.1 * v.2.2 = ROI1(v.1.0, 8, 16) * v.3.0 = v.2.8 * v.3.1 = v.2.1 * v.3.2 = v.2.2 & v.2.3 * v.3.3 = v.2.3 * v.4.0 = v.3.0 * v.4.1 = v.3.1 * v.4.2 = v.3.2 * v.4.2 = v.3.2 * v.4.2 = v.3.2 * v.4.2 = v.3.3 * v.5.0 = v.4.0 * v.5.1 = v.4.1 * v.5.2 = ROI1(v.4.0, 2, 16) * v.5.2 = ROI1(v.4.0, 2, 16) * v.5.2 = v.4.3 * v.6.0 = v.5.0 * v.6.1 = v.5.1 * v.5.2 * v.6.2 = v.5.3 * v.6.0 = v.5.0 * v.6.1 = v.5.1 * v.5.2 * v.6.2 = v.7.3 * v.6.2 = v.7.3 * v.6.2 = v.7.3 * v.6.2 = v.7.3 * v.6.3 = v.7.3 * v.6.4 = v.7.1 * v.6.2 = v.7.2 * v.6.3 = v.7.3 * v.6.0 = v.7.0 * v.0.1 = v.7.1 * v.0.2 = v.7.3 # Output OUT[0] = v.7.0 * v.0.3 = v.7.3 # Coutput OUT[0] = v.7.0 * v.0.3 = v.7.3 * Output OUT[1] = v.7.1 * test implementation TN = [6x0, 0x0] SIMON32_PERM(IN, OUT) print('OUT', str([hex(i) for i in OUT])) </pre>	def ROTR(n, d, bitsize): return ((n >> d) $(n << (bitsize - d))) & (2^{**b})$
<pre>5 # Function implementing the SIMON32_PERM function 6 # Input: 7 # IN: a list of 2 words of 16 bits 7 # OUT: a list of 2 words of 16 bits 7 # Jout 7 # Jout</pre>	
<pre>6 # Input: 7 # IN: a list of 2 words of 16 bits 8 # Output: 9 # OUT: a list of 2 words of 16 bits 10 def SINOU32_PEN(N, OUT): 11 12 13 14 # Input 15 v.0.2 = v.0.3 = 0 15 v.0.2 = v.0.3 = 0 16 # Round function 17 for i in range(10): 18 v.0.2 = v.0.3 19 v.1.1 = v.0.1 10 v.1.2 = ROTL(v.0.0, 1, 16) 10 v.1.3 = v.0.3 10 v.2.0 = v.1.0 11 v.2.2 = v.1.2 12 v.2.3 = ROTL(v.1.0, 8, 16) 12 v.3.0 = v.2.0 13 v.3.0 = v.2.0 14 v.3.2 = v.2.2 & v.2.3 15 v.5.0 = v.4.1 17 v.5.2 = ROTL(v.4.0, 2, 16) 18 v.5.0 = v.5.0 19 v.5.0 = v.5.0 10 v.5.1 = v.5.1 10 v.5.2 = v.5.2 11 v.5.2 = v.5.2 12 v.5.3 = v.4.3 12 v.5.0 = v.5.0 13 v.0.0 = v.5.0 14 v.7.1 = v.6.0 15 v.5.1 = v.7.1 15 v.0.2 = v.7.2 16 v.0.1 = v.7.1 17 v.0.2 = v.7.3 18 Output 19 out[] = v.7.1 19 v.7.3 = v.7.3 19 for i in Out[]) 19 v.7(1) = kool ove] 10 v.5.1 = v.5.1 in IN]))) 10 print('OUT, str([hex(1) for i in IN]))) 10 print('OUT, str([hex(1) for i in OUT])) 10 print('OUT, str([hex(1) for i in OUT])) 10 print('OUT, str([hex(1) for i in OUT]))</pre>	
<pre># ÎN: a list of 2 words of 16 bits # Output: # OUT: a list of 2 words of 16 bits def SINON32_PERM(IN, OUT): # Input v.0.0 = IN[0] v.0.0 = IN[0] v.0.0 = v.0.3 = 0 # Round function for i in range(10): v.1.0 = v.0.0 v.1.0 = v.0.0 v.1.1 = v.0.1 v.1.2 = ROTL(v.0.0, 1, 16) v.1.2 = ROTL(v.0.0, 1, 16) v.1.3 = v.0.3 v.2.0 = v.1.0 v.2.2 = v.1.2 v.2.2 = v.1.2 v.2.2 = v.1.2 v.2.2 = v.1.2 v.2.3 = ROTL(v.1.0, 8, 16) v.3.0 = v.2.0 v.3.1 = v.2.1 v.3.2 = v.2.2 & v.2.3 v.4.0 = v.3.0 v.4.0 = v.3.0 v.4.0 = v.3.0 v.5.1 = v.4.1 v.5.2 = ROTL(v.4.0, 2, 16) v.5.1 = v.4.1 v.5.2 = ROTL(v.4.0, 2, 16) v.5.2 = v.5.2 v.6.3 = v.5.3 v.6.0 = v.5.0 v.6.1 = v.5.1 ^ v.5.2 v.6.2 = v.5.2 t. v.6.2 = v.7.0 v.0.3 = v.7.3 # Output OUT[0] = v.7.0 OUT[0] = v.7.0 OUT[0] = v.7.0 UI[0] = v.7.1 v.0.2 = V.7.3 # foright for i in IN])) print('OUT, str(lhex(i) for i in IN])) print('OUT, str(lhex(i) for i in OUT]))</pre>	
<pre># Output: # Coll: a list of 2 words of 16 bits def SINON32_PERM(IN, OUT): # Input v.0.0 = IN[1] v.0.0 = IN[1] v.0.0 = v.0.0 v.1.0 = v.0.0 v.1.1 = v.0.1 v.1.2 = NOTL(v.0.0, 1, 16) v.1.2 = NOTL(v.0.0, 1, 16) v.1.3 = v.0.3 v.2.0 = v.1.0 v.2.1 = v.1.1 v.2.2 = v.1.2 v.2.3 = NOTL(v.1.0, 8, 16) v.3.0 = v.2.0 v.3.1 = v.2.1 v.3.2 = v.2.2 v.3.3 = v.2.3 v.4.0 = v.3.0 v.4.0 = v.3.0 v.4.1 = v.3.1 v.4.0 = v.3.0 v.4.1 = v.3.1 v.5.2 = NOTL(v.4.0, 2, 16) v.5.3 = v.4.3 v.5.0 = v.5.2 v.5.3 = v.5.3 v.6.0 = v.5.0 v.5.3 = v.5.1 v.5.0 = v.5.1 v.5.0 = v.5.1 v.5.0 = v.5.1 v.5.0 = v.5.2 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.2 = v.5.2 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.2 = v.5.2 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.2 = v.5.2 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.2 = v.5.2 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.2 = v.5.2 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.2 = v.5.2 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.3 = v.5.3 v.5.0 = v.5.0 v.5.0 = v.5.0 v.5</pre>	
<pre>9 # ofr: a list of 2 words of 16 bits 1</pre>	
<pre>10 def SIPON32_PERM(IN, QUT): 11 # Input 12 # Input 13 v.0_0 = IN[0] 14 v.0_1 = IN[1] 15 v.0_2 = v.0_3 = 0 16 17 # Round function 18 for i in range(10): 19 v.1_0 = v.0_0 20 v.1_1 = v.0_1 21 v.1_2 = ROTL(v.0_0, 1, 16) 22 v.1_3 = v.0_3 23 v.2_0 = v.1_0 24 v.2_1 = v.1_1 25 v.2_2 = ROTL(v.1_0, 8, 16) 26 v.3_0 = v.2_0 27 v.3_0 = v.2_0 28 v.3_1 = v.2_1 29 v.3_2 = v.2_2 & v.2_3 30 v.3_3 = v.2_3 31 v.4_0 = v.3_0 32 v.4_1 = v.3_1 ^ v.3_2 33 v.4_0 = v.3_0 34 v.4_3 = v.3_3 35 v.5_0 = v.4_0 36 v.5_1 = v.5_1 ^ v.5_2 47 v.6_2 = v.5_2 48 v.7_2 = v.6_1 49 v.2_1 = v.7_0 40 v.0_1 = v.7_1 41 v.0_2 = v.7_2 51 # Output 52 mOII(0] = v.7_0 53 mOII[0] = v.7_0 54 moint('OUT', str([hex(i) for i in IN]))) 55 moint('OUT', str([hex(i) for i in IN])) 55 moint('OUT', str([hex(i) for i in IN])) 55 moint('OUT', str([hex(i) for i in IN])) 55 moint('OUT', str([hex(i) for i in IN])) 57 moint('OUT', str([hex(i) for i in IN]) 57 moint('OUT', str([hex(i) for i in IN]) 57 moi</pre>	
11 12 13 14 15 15 15 16 16 17 17 18 19 19 10 10 10 10 10 10 10 10 10 10	
12 # Input 13 $v_{0} = IN[0]$ 14 $v_{0} = IN[0]$ 15 $v_{0} = v_{0} = 0$ 17 # Round function 18 for i in range(10): 19 $v_{1} = v_{0} = 0$ 20 $v_{1} = v_{0} = 0$ 21 $v_{1} = v_{0} = 0$ 22 $v_{1} = v_{0} = 0$ 23 $v_{2} = v_{1} = 0$ 24 $v_{2} = v_{1} = 0$ 25 $v_{2} = v_{1} = 0$ 26 $v_{2} = v_{1} = 0$ 27 $v_{2} = v_{1} = 0$ 28 $v_{2} = v_{2} = 0$ 29 $v_{3} = v_{2} = 0$ 20 $v_{3} = v_{2} = 0$ 20 $v_{3} = v_{2} = 0$ 21 $v_{3} = v_{2} = 0$ 22 $v_{2} = v_{1} = 0$ 23 $v_{4} = v_{2} = 0$ 24 $v_{3} = v_{2} = 0$ 25 $v_{4} = v_{2} = 0$ 26 $v_{2} = v_{2} = 0$ 27 $v_{3} = v_{2} = 0$ 28 $v_{4} = v_{3} = 0$ 29 $v_{4} = v_{2} = 0$ 20 $v_{4} = v_{2} = 0$ 20 $v_{4} = v_{3} = 0$ 21 $v_{4} = v_{3} = 0$ 22 $v_{4} = v_{3} = 0$ 23 $v_{4} = v_{3} = 0$ 24 $v_{4} = v_{3} = 0$ 25 $v_{4} = v_{3} = 0$ 26 $v_{5} = v_{4} = 0$ 27 $v_{5} = v_{4} = 0$ 28 $v_{5} = v_{4} = 0$ 29 $v_{5} = v_{5} = 0$ 20 $v_{6} = v_{5} = 0$ 20 $v_{6} = v_{5} = 0$ 21 $v_{6} = v_{5} = 0$ 22 $v_{6} = v_{5} = 0$ 23 $v_{6} = v_{5} = 0$ 24 $v_{6} = v_{5} = 0$ 25 $v_{6} = v_{5} = 0$ 26 $v_{7} = v_{6} = 0$ 27 $v_{6} = v_{5} = 0$ 28 $v_{7} = v_{6} = 0$ 29 $v_{7} = v_{6} = 0$ 20 $v_{7} = v_{7} = 0$ 20 $v_{6} = v_{7} = 0$ 21 $v_{6} = v_{7} = 0$ 22 $v_{7} = v_{6} = 0$ 23 $0UT[0] = v_{7} = 0$ 24 $v_{6} = v_{7} = 0$ 25 $0UT[1] = v_{7} = 0$ 26 $v_{7} = 0$ 27 $v_{6} = 0$ 27 $v_{6} = 0$ 29 $UT[0]$ 20 $UT[0] = v_{7} = 0$ 20 $UT[0] = v_{7} = 0$ 20 $UT[0] = v_{7} = 0$ 20 $UT[0] = v_{7} = 0$ 21 $UTput$ 22 $V(0) = 0$ 23 $UT[0] = v_{7} = 0$ 24 $v_{6} = 0$ 25 $UTOM22 = PEW(UN, 0UT)$ 27 $EW(UN, 0UT)$	def SIMON32_PERM(IN, OUT):
13 $v.0.0 = IN[0]$ 14 $v.0.1 = IN[1]$ 15 $v.0.2 = v.0.3 = 0$ 16 17 # Round function 18 for i in range(10): 19 $v.1.0 = v.0.0$ 20 $v.1.1 = v.0.1$ 21 $v.1.2 = ROTL(v=0, 1, 16)$ 22 $v.1.3 = v.0.3$ 23 $v.2.0 = v.1.0$ 24 $v.2.1 = v.1.1$ 25 $v.2.2 = v.1.2$ 26 $v.2.3 = ROTL(v_1.0, 8, 16)$ 27 $v.3.0 = v.2.0$ 28 $v.3.1 = v.2.1$ 29 $v.3.2 = v.2.2 & v.2.3$ 30 $v.3.3 = v.2.3$ 31 $v.4.0 = v.3.0$ 32 $v.4.1 = v.3.1 + v.3.2$ 33 $v.4.2 = v.3.2$ 34 $v.4.2 = v.3.2$ 35 $v.5.0 = v.4.0$ 36 $v.5.2 = ROTL(v_4.0, 2, 16)$ 37 $v.5.2 = ROTL(v_4.0, 2, 16)$ 38 $v.5.3 = v.4.3$ 39 $v.6.0 = v.5.0$ 40 $v.6.1 = v.5.1 + v.5.2$ 41 $v.6.2 = v.5.2$ 42 $v.6.3 = v.5.3$ 43 $v.7.0 = v.6.1$ 44 $v.7.1 = v.7.1$ 55 f function 51 $N = [0x0, 0x0]$ 52 $IVTOP$ 53 $OUT[0] = v.7.0$ 54 $v.100T[0] = v.7.0$ 55 $INON32_PERM(IN, OUT)$ 56 $VTOP$ is in IN]))) 57 $IVT('OUT', str((IneX(I) for i in IN])))$ 57 $IVT('OUT', str((IneX(I) for i in IN])))$	
13 $v.0.0 = IN[0]$ 14 $v.0.1 = IN[1]$ 15 $v.0.2 = v.0.3 = 0$ 16 17 # Round function 18 for i in range(10): 19 $v.1.0 = v.0.0$ 20 $v.1.1 = v.0.1$ 21 $v.1.2 = ROTL(v=0, 1, 16)$ 22 $v.1.3 = v.0.3$ 23 $v.2.0 = v.1.0$ 24 $v.2.1 = v.1.1$ 25 $v.2.2 = v.1.2$ 26 $v.2.3 = ROTL(v_1.0, 8, 16)$ 27 $v.3.0 = v.2.0$ 28 $v.3.1 = v.2.1$ 29 $v.3.2 = v.2.2 & v.2.3$ 30 $v.3.3 = v.2.3$ 31 $v.4.0 = v.3.0$ 32 $v.4.1 = v.3.1 + v.3.2$ 33 $v.4.2 = v.3.2$ 34 $v.4.2 = v.3.2$ 35 $v.5.0 = v.4.0$ 36 $v.5.2 = ROTL(v_4.0, 2, 16)$ 37 $v.5.2 = ROTL(v_4.0, 2, 16)$ 38 $v.5.3 = v.4.3$ 39 $v.6.0 = v.5.0$ 40 $v.6.1 = v.5.1 + v.5.2$ 41 $v.6.2 = v.5.2$ 42 $v.6.3 = v.5.3$ 43 $v.7.0 = v.6.1$ 44 $v.7.1 = v.7.1$ 55 f function 51 $N = [0x0, 0x0]$ 52 $IVTOP$ 53 $OUT[0] = v.7.0$ 54 $v.100T[0] = v.7.0$ 55 $INON32_PERM(IN, OUT)$ 56 $VTOP$ is in IN]))) 57 $IVT('OUT', str((IneX(I) for i in IN])))$ 57 $IVT('OUT', str((IneX(I) for i in IN])))$	
14 $v.0.1 = IM[1]$ 15 $v.0.2 = v.0.3 = 0$ 17 # Round function 18 for i in range(10): 19 $v.1.0 = v.0.0$ 20 $v.1.1 = v.0.1$ 21 $v.1.2 = ROTL(v.0.0, 1, 16)$ 22 $v.1.3 = v.0.3$ 23 $v.2.0 = v.1.0$ 24 $v.2.1 = v.1.1$ 25 $v.2.2 = v.1.2$ 26 $v.2.3 = ROTL(v.1.0, 8, 16)$ 27 $v.3.0 = v.2.3$ 28 $v.2.3 = v.2.3$ 30 $v.3.3 = v.2.3$ 31 $v.4.0 = v.3.0$ 32 $v.4.1 = v.3.1 + v.3.2$ 33 $v.4.2 = v.3.2$ 34 $v.4.3 = v.3.3$ 35 $v.5.0 = v.4.0$ 36 $v.5.1 = v.4.1$ 37 $v.5.2 = ROTL(v.4.0, 2, 16)$ 38 $v.5.3 = v.4.3$ 39 $v.6.0 = v.5.0$ 40 $v.6.1 = v.5.1 + v.5.2$ 41 $v.6.2 = v.5.3$ 42 $v.0.3 = v.7.3$ 43 $v.7.0 = v.6.1$ 44 $v.7.1 = v.7.1$ 45 $v.0.2 = v.7.2$ 46 $v.7.2 = v.6.2$ 47 $v.0.0 = v.7.0$ 48 $v.0.1 = v.7.1$ 49 $v.0.2 = v.7.2$ 49 $v.0.3 = v.7.3$ 41 $v.0.2 = v.7.2$ 51 $v.0.11 = v.7.1$ 52 $v.0.3 = v.7.3$ 53 $OUT[0] = v.7.0$ 54 $OUT[0] = v.7.0$ 55 $IMOM32_PERM(IN, OUT)$ 56 $IIMOM32_PERM(IN, OUT)$ 57 $IIM = [0x0, 0x0]$ 50 $OUT = [0x0, 0x0]$ 51 $OUT(V)$, $str((hex(i)) for i in OUT]))$	
15 $v_{.0} = v_{.0} = 0$ 17 # Round function 18 for i in range(10): 19 $v_{.1,0} = v_{.0,0}$ 20 $v_{.1,1} = v_{.0,1}$ 21 $v_{.1,2} = ROTL(v_{.0,0,1,16})$ 22 $v_{.1,3} = v_{.0,3}$ 23 $v_{.2,0} = v_{.1,0}$ 24 $v_{.2,1} = v_{.1,1}$ 25 $v_{.2,2} = ROTL(v_{.1,0,8,16})$ 26 $v_{.2,3} = ROTL(v_{.1,0,8,16})$ 27 $v_{.3,0} = v_{.2,0}$ 28 $v_{.3,1} = v_{.2,1}$ 29 $v_{.3,2} = v_{.2,2} 8 v_{.2,3}$ 30 $v_{.3,3} = v_{.2,3}$ 31 $v_{.4,0} = v_{.3,0}$ 32 $v_{.4,1} = v_{.3,1} \wedge v_{.3,2}$ 33 $v_{.4,2} = v_{.3,2}$ 34 $v_{.4,3} = v_{.3,3}$ 35 $v_{.5,0} = v_{.4,0}$ 36 $v_{.5,1} = v_{.4,1}$ 37 $v_{.5,2} = ROTL(v_{.4,0,2,16})$ 38 $v_{.5,3} = v_{.4,3}$ 39 $v_{.6,0} = v_{.5,0}$ 40 $v_{.6,1} = v_{.5,1} \wedge v_{.5,2}$ 41 $v_{.6,2} = v_{.5,2}$ 42 $v_{.6,3} = v_{.5,3}$ 43 $v_{.7,2} = v_{.6,2}$ 44 $v_{.7,2} = v_{.6,2}$ 45 $v_{.7,2} = v_{.6,2}$ 46 $v_{.7,2} = v_{.6,2}$ 47 $v_{.0,0} = v_{.7,0}$ 48 $v_{.0,1} = v_{.7,1}$ 49 $v_{.0,2} = v_{.7,2}$ 50 $v_{.0,3} = v_{.7,3}$ 51 # Output 52 $mOT[0] = v_{.7,0}$ 53 $OOT[0] = v_{.7,0}$ 54 $motion$ 55 $mOT[0] = v_{.7,0}$ 56 $motion$ 57 $motion$ 58 $motion$ 59 $motion$ 51 $motion$ 50 $motion$ 51 $motion$ 51 $motion$ 51 $motion$ 51 $motion$ 51 $motion$ 51 $motion$ 52 $motion$ 53 $motion$ 54 $motion$ 55 $motion$ 56 $motion$ 57 $motion$ 57 $motion$ 57 $motion$ 50 m	$\sqrt{61} - 100$
16 Formula Formula Formula 17 # Round function 18 for i in range(10): 19 v _1_0 = v_0.0 20 v _1_1 = v_0.1 21 v _1_2 = ROTL(v_0_0, 1, 16) 22 v _1_3 = v_0.3 23 v _2_0 = v_1_0 24 v _2_1 = v_1_1 25 v _2_2 = v_1_2 26 v _2_3 = ROTL(v_1_0, 8, 16) 27 v _3_0 = v_2_0 28 v _3_1 = v_2_1 29 v _3_2 = v_2_2 & v_2_3 30 v _3_3 = v_2_3 31 v _4_0 = v_3_0 22 v _4_1 = v_3_1 ^ v_3_2 33 v _4_2 = v_3_2 34 v _4_3 = v_3_3 35 v _5_0 = v_4_0 36 v _5_1 = v_4_1 37 v _5_2 = ROTL(v_4_0, 2, 16) 38 v _5_3 = v_4_3 39 v _6_0 = v_5_0 40 v _6_1 = v_5_1 ^ v_5_2 41 v _6_2 = v_5_2 42 v _6_3 = v_5_3 43 v _7_0 = v_6_1 44 v _7_1 = v_6_0 45 v _7_2 = v_6_2 46 v _7_3 v _6_3 47 v _0_0 = v_7_0 48 v _0_1 = v_7_1 49 v _0_3 = v_7_3 47 t _st implementation 50 UI [0] = v_7_1 51 f t _st(lhex(i) for i in IN])) 51 f i (OUT, str(lhex(i) for i in OUT]))	
17 # Round function 18 for i in range(10): 19 $v_{\perp}1_{\perp} = v_{\cdot}0_{\perp}0$ 20 $v_{\perp}1_{\perp} = v_{\cdot}0_{\perp}1$ 21 $v_{\perp}2_{\perp} = ROTL(v_{\bullet}0_{\bullet}0, 1, 16)$ 22 $v_{\perp}1_{\perp} = v_{\cdot}1_{\perp}1$ 23 $v_{\perp}2_{\cdot}0 = v_{\perp}1_{\cdot}0$ 24 $v_{\perp}2_{\perp} = v_{\perp}1_{\perp}1$ 25 $v_{\perp}2_{\perp} = v_{\perp}1_{\perp}2$ 26 $v_{\perp}3_{\perp} = v_{\perp}2_{\perp}1$ 27 $v_{\perp}3_{\perp} = v_{\perp}2_{\perp}1$ 28 $v_{\perp}3_{\perp} = v_{\perp}2_{\perp}1$ 29 $v_{\perp}3_{\perp} = v_{\perp}2_{\perp}3$ 30 $v_{\perp}3_{\perp} = v_{\perp}2_{\perp}3$ 31 $v_{\perp}4_{\perp} = v_{\perp}3_{\perp}^{\perp}1 \wedge v_{\perp}3_{\perp}2$ $v_{\perp}4_{\perp} = v_{\perp}3_{\perp}^{\perp}1 \wedge v_{\perp}3_{\perp}2$ $v_{\perp}4_{\perp} = v_{\perp}3_{\perp}^{\perp}1 \wedge v_{\perp}3_{\perp}2$ $v_{\perp}5_{\perp} = v_{\perp}3_{\perp}3$ $v_{\perp}4_{\perp} = v_{\perp}3_{\perp}^{\perp}1 \wedge v_{\perp}3_{\perp}2$ $v_{\perp}5_{\perp} = v_{\perp}3_{\perp}3$	$\mathbf{v}_{-}\mathbf{v}_{-}\mathbf{z} = \mathbf{v}_{-}\mathbf{v}_{-}\mathbf{z} = \mathbf{v}_{-}\mathbf{v}_{-}\mathbf{z}$
$ \begin{array}{c c c c c c c c c c c c c c c c c c c $	
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	
20 $\sqrt{-1} = \sqrt{-1}$ 21 $\sqrt{-1} = \sqrt{-1}$ 21 $\sqrt{-1} = \sqrt{-1}$ 22 $\sqrt{-1} = \sqrt{-1}$ 23 $\sqrt{-2} = \sqrt{-1}$ 24 $\sqrt{-2} = \sqrt{-1}$ 25 $\sqrt{-2} = \sqrt{-1}$ 26 $\sqrt{-2} = \sqrt{-1}$ 27 $\sqrt{-3} = \sqrt{-2}$ 28 $\sqrt{-3} = \sqrt{-2}$ 29 $\sqrt{-3} = \sqrt{-2}$ 29 $\sqrt{-3} = \sqrt{-2}$ 20 $\sqrt{-3} = \sqrt{-2}$ 20 $\sqrt{-3} = \sqrt{-2}$ 20 $\sqrt{-3} = \sqrt{-2}$ 21 $\sqrt{-3} = \sqrt{-2}$ 22 $\sqrt{-3} = \sqrt{-2}$ 23 $\sqrt{-4} = \sqrt{-3}$ 24 $\sqrt{-4} = \sqrt{-3}$ 25 $\sqrt{-4} = \sqrt{-3}$ 26 $\sqrt{-5} = \sqrt{-4}$ 27 $\sqrt{-5} = \sqrt{-4}$ 28 $\sqrt{-3} = \sqrt{-3}$ 29 $\sqrt{-4} = \sqrt{-3}$ 20 $\sqrt{-4} = \sqrt{-3}$ 20 $\sqrt{-4} = \sqrt{-3}$ 21 $\sqrt{-4} = \sqrt{-3}$ 22 $\sqrt{-4} = \sqrt{-3}$ 23 $\sqrt{-4} = \sqrt{-3}$ 24 $\sqrt{-4} = \sqrt{-3}$ 25 $\sqrt{-5} = \sqrt{-4}$ 27 $\sqrt{-5} = \sqrt{-5}$ 28 $\sqrt{-5} = \sqrt{-5}$ 29 $\sqrt{-6} = \sqrt{-5}$ 20 $\sqrt{-6} = \sqrt{-5}$ 20 $\sqrt{-6} = \sqrt{-5}$ 21 $\sqrt{-6} = \sqrt{-5}$ 22 $\sqrt{-6} = \sqrt{-7}$ 23 $\sqrt{-6} = \sqrt{-5}$ 24 $\sqrt{-6} = \sqrt{-7}$ 25 $\sqrt{-6} = \sqrt{-7}$ 26 $\sqrt{-7} = \sqrt{-6}$ 27 $\sqrt{-6} = \sqrt{-7}$ 29 $\sqrt{-6} = \sqrt{-7}$ 20 $\sqrt{-6} = \sqrt{-7}$ 30 $\sqrt{-6} = \sqrt{-7}$ 31 $\sqrt{-6} = \sqrt{-7}$ 32 $\sqrt{-6} = \sqrt{-7}$ 33 $\sqrt{-6} = \sqrt{-7}$ 34 $\sqrt{-6} = \sqrt{-7}$ 35 $\sqrt{-6} = \sqrt{-7}$ 36 $\sqrt{-7} = \sqrt{-7}$ 37 $\sqrt{-6} = \sqrt{-7}$ 39 $\sqrt{-6} = \sqrt{-7}$ 30 $\sqrt{-6} = \sqrt{-7}$ 30 $\sqrt{-6} = \sqrt{-7}$ 31 $\sqrt{-6} = \sqrt{-7}$ 32 $\sqrt{-6} = \sqrt{-7}$ 33 $\sqrt{-6} = \sqrt{-7}$ 34 $\sqrt{-7} = \sqrt{-6}$ 35 $\sqrt{-6} = \sqrt{-7}$ 36 $\sqrt{-7} = \sqrt{-7}$ 37 $\sqrt{-6} = \sqrt{-7}$ 39 $\sqrt{-6} = \sqrt{-7}$ 30 $\sqrt{-6} = \sqrt{-7}$ 30 $\sqrt{-6} = \sqrt{-7}$ 31 $\sqrt{-6} = \sqrt{-7}$ 32 $\sqrt{-6} = \sqrt{-7}$ 33 $\sqrt{-6} = \sqrt{-7}$ 34 $\sqrt{-7} = \sqrt{-7}$ 35 $\sqrt{-6} = \sqrt{-7}$ 36 $\sqrt{-7} = \sqrt{-7}$ 37 $\sqrt{-6} = \sqrt{-7}$ 39 $\sqrt{-6} = \sqrt{-7}$ 30 $\sqrt{-6} = \sqrt{-7}$ 31 $\sqrt{-6} = \sqrt{-7}$ 32 $\sqrt{-6} = \sqrt{-7}$ 33 $\sqrt{-6} = \sqrt{-7}$ 34 $\sqrt{-7} = \sqrt{-7}$ 35 $\sqrt{-6} = \sqrt{-7}$ 37 $\sqrt{-6} = \sqrt{-7}$ 38 $\sqrt{-6} = \sqrt{-7}$ 39 $\sqrt{-6} = \sqrt{-7}$ 30 \sqrt	
20 21 22 23 24 24 25 25 26 27 27 27 27 27 27 27 27 27 27	v_1_0 = v_0_0
22 $\sqrt{13} = \sqrt{0.3}$ 23 $\sqrt{2.0} = \sqrt{1.0}$ 24 $\sqrt{2.1} = \sqrt{1.1}$ 25 $\sqrt{2.2} = \sqrt{1.2}$ 26 $\sqrt{2.3} = 80TL(v.1.0, 8, 16)$ 27 $\sqrt{3.0} = \sqrt{2.0}$ 28 $\sqrt{3.1} = \sqrt{2.1}$ 29 $\sqrt{3.2} = \sqrt{2.2}$ 20 $\sqrt{3.2} = \sqrt{2.2}$ 30 $\sqrt{3.3} = \sqrt{2.3}$ 31 $\sqrt{4.0} = \sqrt{3.0}$ 32 $\sqrt{4.1} = \sqrt{3.1} + \sqrt{3.2}$ 33 $\sqrt{4.2} = \sqrt{3.2}$ 34 $\sqrt{4.3} = \sqrt{3.3}$ 35 $\sqrt{5.0} = \sqrt{4.0}$ 36 $\sqrt{5.1} = \sqrt{4.1}$ 37 $\sqrt{5.2} = ROTL(v.4.0, 2, 16)$ 38 $\sqrt{5.2} = \sqrt{5.2}$ 40 $\sqrt{6.1} = \sqrt{5.1} + \sqrt{5.2}$ 41 $\sqrt{6.2} = \sqrt{5.2}$ 42 $\sqrt{6.3} = \sqrt{5.3}$ 43 $\sqrt{7.0} = \sqrt{6.0}$ 44 $\sqrt{7.1} = \sqrt{6.0}$ 45 $\sqrt{7.2} = \sqrt{6.2}$ 46 $\sqrt{7.3} = \sqrt{7.0}$ 47 $\sqrt{0.0} = \sqrt{7.0}$ 48 $\sqrt{0.1} = \sqrt{7.1}$ 59 $\sqrt{0.10} = \sqrt{7.0}$ 50 $\sqrt{10} = \sqrt{7.2}$ 50 $\sqrt{10} = \sqrt{7.2}$ 51 $\sqrt{10} = \sqrt{7.2}$ 52 $\frac{\#}{0} Out[0] = \sqrt{7.0}$ 53 $OUt[0] = \sqrt{7.2}$ 54 $\sqrt{10} = \sqrt{7.1}$ 55 $\sqrt{10} = \sqrt{7.0}$ 56 $\sqrt{10} = \sqrt{7.0}$ 57 $\sqrt{10} = \sqrt{7.0}$ 58 $\sqrt{10} = \sqrt{7.1}$ 59 $\sqrt{10} = \sqrt{7.0}$ 50 $\sqrt{10} = \sqrt{7.0}$ 51 $\sqrt{10} = \sqrt{7.1}$ 52 $\frac{\#}{10} Cutput$ 53 $OUt[1] = \sqrt{7.1}$ 54 $\sqrt{10} = \sqrt{10} (V.1, Svr([hex(1) for i in IN]))$ 55 $\sqrt{10} = [6x0, 0x0]$ 57 $\sqrt{10} + \sqrt{10} (V.1, str([hex(1) for i in OUT]))$	$v_1 = v_0 = 1$
22 $v \cdot 1.3 = v \cdot 0.3$ 23 $v \cdot 2.0 = v \cdot 1.0$ 24 $v \cdot 2.1 = v \cdot 1.1$ 25 $v \cdot 2.2 = v \cdot 1.2$ 26 $v \cdot 2.3 = ROTL(v \cdot 1.0, 8, 16)$ 27 $v \cdot 3.0 = v \cdot 2.0$ 28 $v \cdot 3.1 = v \cdot 2.1$ 29 $v \cdot 3.2 = v \cdot 2.2 \ 8 \cdot v \cdot 2.3$ 30 $v \cdot 3.3 = v \cdot 2.3$ 31 $v \cdot 4.0 = v \cdot 3.0$ 32 $v \cdot 4.1 = v \cdot 3.1 \land v \cdot 3.2$ 33 $v \cdot 4.2 = v \cdot 3.2$ 34 $v \cdot 4.3 = v \cdot 3.3$ 35 $v \cdot 5.0 = v \cdot 4.0$ 36 $v \cdot 5.1 = v \cdot 4.1$ 37 $v \cdot 5.2 = ROTL(v \cdot 4.0, 2, 16)$ 38 $v \cdot 5.3 = v \cdot 4.3$ 39 $v \cdot 6.0 = v \cdot 5.0$ 40 $v \cdot 6.1 = v \cdot 5.1 \land v \cdot 5.2$ 41 $v \cdot 6.2 = v \cdot 5.2$ 42 $v \cdot 6.3 = v \cdot 5.3$ 43 $v \cdot 7.0 = v \cdot 6.0$ 44 $v \cdot 7.1 = v \cdot 6.0$ 45 $v \cdot 7.2 = v \cdot 6.2$ 46 $v \cdot 7.3 = v \cdot 7.3$ 47 $v \cdot 0.0 = v \cdot 7.0$ 48 $v \cdot 0.1 = v \cdot 7.1$ 49 $v \cdot 0.2 = v \cdot 7.2$ 50 $v \cdot 0.3 = v \cdot 7.3$ 51 $\# Output$ 52 $\# Output$ 53 $OUT[0] = v \cdot 7.0$ 54 $OUT[0] = v \cdot 7.0$ 55 $OUT[1] = v \cdot 7.1$ 56 $\# test implementation$ 57 $IN = [0x0, 0x0]$ 59 $SIMOM32 - PERM(N, 0UT)$ 60 $print(`OUT', str([hex(i) for i in IN])))$ 61 $print(`OUT', str([hex(i) for i in OUT]))$	$v_1_2 = ROTL(v_0_0, 1, 16)$
23 $\sqrt{-20} = \sqrt{-10}$ 24 $\sqrt{-20} = \sqrt{-10}$ 25 $\sqrt{-2} = \sqrt{-10}$ 26 $\sqrt{-2} = \sqrt{-11}$ 27 $\sqrt{-3} = \sqrt{-11}$ 28 $\sqrt{-3} = \sqrt{-2}$ 29 $\sqrt{-3} = \sqrt{-2}$ 29 $\sqrt{-3} = \sqrt{-2}$ 20 $\sqrt{-3} = \sqrt{-2}$ 20 $\sqrt{-3} = \sqrt{-2}$ 20 $\sqrt{-3} = \sqrt{-2}$ 20 $\sqrt{-3} = \sqrt{-2}$ 21 $\sqrt{-3} = \sqrt{-2}$ 22 $\sqrt{-3} = \sqrt{-2}$ 23 $\sqrt{-3} = \sqrt{-2}$ 24 $\sqrt{-3} = \sqrt{-3}$ 25 $\sqrt{-50} = \sqrt{-40}$ 27 $\sqrt{-5} = \sqrt{-40}$ 27 $\sqrt{-5} = \sqrt{-40}$ 28 $\sqrt{-5} = \sqrt{-40}$ 29 $\sqrt{-5} = \sqrt{-40}$ 20 $\sqrt{-5} = \sqrt{-40}$ 20 $\sqrt{-5} = \sqrt{-40}$ 21 $\sqrt{-5} = \sqrt{-5}$ 21 $\sqrt{-5} = \sqrt{-5}$ 22 $\sqrt{-5} = \sqrt{-5}$ 23 $\sqrt{-5} = \sqrt{-5}$ 24 $\sqrt{-5} = \sqrt{-5}$ 25 $\sqrt{-5} = \sqrt{-2}$ 26 $\sqrt{-7} = \sqrt{-5}$ 27 $\sqrt{-5} = \sqrt{-2}$ 28 $\sqrt{-5} = \sqrt{-2}$ 29 $\sqrt{-2} = \sqrt{-2}$ 20 $\sqrt{-2} = \sqrt{-2}$ 20 $\sqrt{-2} = \sqrt{-2}$ 20 $\sqrt{-2} = \sqrt{-2}$ 21 $\sqrt{-2} = \sqrt{-2}$ 22 $\sqrt{-2} = \sqrt{-2}$ 23 $\sqrt{-3} = \sqrt{-3}$ 24 $\sqrt{-3} = \sqrt{-2}$ 25 $\sqrt{-3} = \sqrt{-2}$ 26 $\sqrt{-3} = \sqrt{-2}$ 27 $\sqrt{-3} = \sqrt{-2}$ 28 $\sqrt{-3} = \sqrt{-2}$ 29 $\sqrt{-3} = \sqrt{-2}$ 20 $\sqrt{-3} = \sqrt{-2}$ 30 $\sqrt{-3} = \sqrt{-2}$ 31 $\sqrt{-3} = \sqrt{-3}$ 32 $\sqrt{-3} = \sqrt{-3}$ 33 $\sqrt{-3} = \sqrt{-3}$ 34 $\sqrt{-3} = \sqrt{-3}$ 35 $\sqrt{-3} = \sqrt{-3}$ 35 $\sqrt{-3} = \sqrt{-3}$ 36 $\sqrt{-3} = \sqrt{-3}$ 37 $\sqrt{-3} = \sqrt{-3}$ 39 $\sqrt{-3} = \sqrt{-3}$ 30 $\sqrt{-3} = \sqrt{-3}$ 30 $\sqrt{-3} = \sqrt{-3}$ 31 $\sqrt{-3} = \sqrt{-3}$ 32 $\sqrt{-3} = \sqrt{-3}$ 33 $\sqrt{-3} = \sqrt{-3}$ 34 $\sqrt{-3} = \sqrt{-3}$ 35 $\sqrt{-3} = \sqrt{-3}$ 36 $\sqrt{-3} = \sqrt{-3}$ 37 $\sqrt{-3} = \sqrt{-3}$ 38 $\sqrt{-3} = \sqrt{-3}$ 39 $\sqrt{-3} = \sqrt{-3}$ 30 $\sqrt{-3} = \sqrt{-3}$ 31 $\sqrt{-3} = \sqrt{-3}$ 31 $\sqrt{-3} = \sqrt{-3}$ 32 $\sqrt{-3} = \sqrt{-3}$ 33 $\sqrt{-3} = \sqrt{-3}$ 34 $\sqrt{-3} = \sqrt{-3}$ 35 $\sqrt{-3} = \sqrt{-3}$ 36 $\sqrt{-3} = \sqrt{-3}$ 37 $\sqrt{-3} = \sqrt{-3}$ 38 $\sqrt{-3} = \sqrt{-3}$ 39 $\sqrt{-3} = \sqrt{-3}$ 30 $\sqrt{-3} = \sqrt{-3}$ 30 $\sqrt{-3} = \sqrt{-3}$ 30 $\sqrt{-3} = \sqrt{-3}$ 30 $\sqrt{-3} = \sqrt{-3}$ 31 $\sqrt{-3} = \sqrt{-3}$ 31 $\sqrt{-3} = \sqrt{-3}$ 32 $\sqrt{-3} = \sqrt{-3}$ 33 $\sqrt{-3} = \sqrt{-3}$ 34 $\sqrt{-3} = \sqrt{-3}$ 35 $\sqrt{-3} = \sqrt{-3}$ 36 $\sqrt{-3} = \sqrt{-3}$ 37 $\sqrt{-3} = \sqrt{-3}$ 38 $\sqrt{-3} = \sqrt{-3}$ 39 $\sqrt{-3} = \sqrt{-3}$ 30 $\sqrt{-3} = \sqrt{-3}$ 30 $\sqrt{-3} = \sqrt{-3}$ 30 $\sqrt{-3} = \sqrt{-3}$ 31 $\sqrt{-3} = \sqrt{-3}$ 31 $\sqrt{-3} = \sqrt{-3}$ 31 $\sqrt{-3} $	
24 $\sqrt{2}1 = \sqrt{2}1$ 25 $\sqrt{2}2 = \sqrt{2}12$ 26 $\sqrt{2}2 = \sqrt{2}2$ 27 $\sqrt{3}0 = \sqrt{2}0$ 28 $\sqrt{3}1 = \sqrt{2}1$ 29 $\sqrt{3}2 = \sqrt{2}2 & \sqrt{2}3$ 30 $\sqrt{3}3 = \sqrt{2}3$ 31 $\sqrt{4}0 = \sqrt{3}0$ 32 $\sqrt{4}1 = \sqrt{3}1 & \sqrt{3}2$ 33 $\sqrt{4}2 = \sqrt{3}2$ 34 $\sqrt{4}3 = \sqrt{3}0$ 35 $\sqrt{5}0 = \sqrt{4}0$ 36 $\sqrt{5}1 = \sqrt{4}1$ 37 $\sqrt{5}2 = ROT((\sqrt{4}0, 2, 16))$ 38 $\sqrt{5}3 = \sqrt{4}3$ 39 $\sqrt{6}0 = \sqrt{5}0$ 40 $\sqrt{6}1 = \sqrt{5}1 & \sqrt{5}2$ 41 $\sqrt{6}2 = \sqrt{5}2$ 42 $\sqrt{6}3 = \sqrt{5}3$ 43 $\sqrt{7}0 = \sqrt{6}0$ 44 $\sqrt{7}1 = \sqrt{6}0$ 45 $\sqrt{7}2 = \sqrt{6}2$ 46 $\sqrt{7}3 = \sqrt{6}3$ 47 $\sqrt{9}0 = \sqrt{7}0$ 48 $\sqrt{9}1 = \sqrt{7}1$ 49 $\sqrt{9}0 = \sqrt{7}0$ 49 $\sqrt{9}0 = \sqrt{7}0$ 40 $\sqrt{6}0 = \sqrt{7}0$ 41 $\sqrt{6}0 = \sqrt{7}0$ 42 $\sqrt{6}3 = \sqrt{7}3$ 43 $\sqrt{7}0 = \sqrt{7}0$ 44 $\sqrt{7}1 = \sqrt{6}0$ 45 $\sqrt{7}2 = \sqrt{6}2$ 47 $\sqrt{9}0 = \sqrt{7}0$ 48 $\sqrt{9}1 = \sqrt{7}1$ 49 $\sqrt{9}2 = \sqrt{7}2$ 50 $\sqrt{9}3 = \sqrt{7}3$ 51 $\frac{1}{1}$ 52 $\frac{1}{1}$ Output 53 OUT[0] = $\sqrt{7}0$ 54 OUT[1] = $\sqrt{7}1$ 55 $\frac{1}{1}$ FEW(IN, OUT) 56 $\frac{1}{1}$ FIN([hex(i) for i in IN])) 57 IN = [0x0, 0x0] 59 SIMON32 PERM(IN, OUT) 50 $\frac{1}{1}$ $\sqrt{1}$ (Like(i) for i in IN]))	
25 $\sqrt{-2} = \sqrt{-2} = \sqrt{-2}$ 26 $\sqrt{-2} = \sqrt{-2} = \sqrt{-2}$ 27 $\sqrt{-3} = 80TL(v_1, 0, 8, 16)$ 27 $\sqrt{-3}, 0 = v_2, 0$ 28 $\sqrt{-3}, 1 = v_2, 1$ 29 $\sqrt{-3}, 2 = v_2, 2, 8, v_2, 3$ 30 $\sqrt{-3}, 3 = v_2, 3$ 31 $\sqrt{-4}, 0 = v_3, 0$ 32 $\sqrt{-4}, 1 = v_3, 1, \sqrt{-3}, 2$ 33 $\sqrt{-4}, 2 = v_3, 2$ 34 $\sqrt{-4}, 2 = v_3, 2$ 35 $\sqrt{-5}, 0 = \sqrt{-4}, 0$ 36 $\sqrt{-5}, 2 = ROTL(v_4, 0, 2, 16)$ 37 $\sqrt{-5}, 2 = ROTL(v_4, 0, 2, 16)$ 38 $\sqrt{-5}, 3 = v_4, 3$ 39 $\sqrt{-6}, 0 = \sqrt{-5}, 0$ 40 $\sqrt{-6}, 1 = \sqrt{-5}, 1, \sqrt{-5}, 2$ 41 $\sqrt{-6}, 2 = \sqrt{-5}, 2$ 42 $\sqrt{-6}, 3 = \sqrt{-5}, 3$ 43 $\sqrt{-7}, 0 = \sqrt{-6}, 1$ 44 $\sqrt{-7}, 1 = \sqrt{-6}, 0$ 45 $\sqrt{-7}, 2 = \sqrt{-6}, 2$ 46 $\sqrt{-7}, 3 = \sqrt{-6}, 3$ 47 $\sqrt{-0}, 0 = \sqrt{-7}, 0$ 48 $\sqrt{-0}, 1 = \sqrt{-7}, 1$ 49 $\sqrt{-0}, 0 = \sqrt{-7}, 0$ 40 $\sqrt{-0}, 0 = \sqrt{-7}, 0$ 41 $\sqrt{-0}, 0 = \sqrt{-7}, 0$ 42 $\sqrt{-3}, 3 = \sqrt{-7}, 3$ 53 $0 \text{ UT[0]} = \sqrt{-7}, 0$ 54 $0 \text{ UT[1]} = \sqrt{-7}, 1$ 55 $\# \text{ test implementation}$ 57 $\text{IN} = [0x0, 0x0]$ 59 $\text{ SIMOM32}, \text{PERM(IN, 0UT)}$ 59 $\text{ SIMOM32}, \text{PERM(IN, OUT)}$ 50 $ print('UV', str([hex(1) for i in UN])))$ 61 $ print('UV', str([hex(1) for i in 0UT])))$	
26 $\sqrt{-2} = ROT[(v_1,0, 8, 16)$ 27 $v_3,0 = v_2,0$ 28 $v_3,1 = v_2,1$ 29 $v_3,2 = v_2,2,8 v_2,3$ 30 $v_3,3 = v_2,3$ 31 $v_4,0 = v_3,0$ 32 $v_4,1 = v_3,1 + v_3,2$ 33 $v_4,2 = v_3,2$ 34 $v_4,3 = v_3,3$ 35 $v_5,0 = v_4,0$ 36 $v_5,1 = v_4,1$ 37 $v_5,2 = ROT[(v_4,0,2,16)$ 38 $v_5,3 = v_4,3$ 39 $v_6,0 = v_5,0$ 40 $v_6,1 = v_5,1 + v_5,2$ 41 $v_6,2 = v_5,2$ 42 $v_6,3 = v_5,3$ 43 $v_7,0 = v_6,1$ 44 $v_7,1 = v_6,0$ 45 $v_7,2 = v_6,2$ 46 $v_7,3 = v_6,3$ 47 $v_0,0 = v_7,0$ 48 $v_0,1 = v_7,1$ 49 $v_0,2 = v_7,2$ 50 $v_0,3 = v_7,3$ 51 # Output 52 # Output 53 OUT[0] = v_7,1 54 $v_1 = [0x0, 0x0]$ 55 $m(0,0,0x0]$ 57 $IN = [0x0, 0x0]$ 59 $SIMON32,PERM(IN, OUT)$ 50 $r_1(V,V,str([hex(i) for i in IN])))$ 60 $print((U,V,str([hex(i) for i in OUT])))$	
27 $ $ $v_3 = v_2 = v_2 = 0$ 28 $v_3 = v_2 = v_2 = 8 v_2 = 3$ 30 $v_3 = v_2 = v_2 = 8 v_2 = 3$ 31 $v_4 = v_3 = v_2 = 3$ 32 $v_4 = v_3 = v_3 = 3$ 33 $v_4 = v_3 = v_3 = 3$ 34 $v_4 = v_3 = v_3 = 3$ 35 $v_5 = v_4 = v_3 = 3$ 36 $v_5 = v_4 = v_3 = 1$ 37 $v_5 = s = s = 1$ 39 $v_5 = s = v_4 = 1$ 39 $v_5 = s = v_4 = 1$ 39 $v_5 = s = v_4 = 1$ 40 $v_5 = v_5 = s = 1$ 40 $v_5 = v_5 = s = 1$ 41 $v_5 = v_5 = s = 1$ 42 $v_6 = v_5 = s = 1$ 43 $v_7 = v_5 = s = 1$ 44 $v_7 = v_5 = s = 1$ 45 $v_7 = v_5 = s = 1$ 46 $v_7 = v_5 = s = 1$ 47 $v_9 = s = v_7 = 1$ 48 $v_9 = v_7 = v_6 = 1$ 49 $v_9 = v_7 = s = 1$ 50 $v_9 = v_7 = 1$ 51 f Output 52 f Out[1] = $v_7 = 1$ 53 OUT[2] = $v_7 = 1$ 54 f test implementation 57 IN = [sso, sso] 59 SIMON32 = PERM(IN, OUT) 60 print('OUT', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))	
28 $v_{3,1} = v_{2,1}$ 29 $v_{3,2} = v_{2,2} \& v_{2,3}$ 30 $v_{3,3} = v_{2,3}$ 31 $v_{4,0} = v_{3,0}$ 32 $v_{4,1} = v_{3,1} \wedge v_{3,2}$ 33 $v_{4,2} = v_{3,2}$ 34 $v_{4,3} = v_{3,3}$ 35 $v_{5,0} = v_{4,0}$ 36 $v_{5,1} = v_{4,1}$ 37 $v_{5,2} = ROTL(v_{4,0}, 2, 16)$ 38 $v_{5,3} = v_{4,3}$ 39 $v_{6,0} = v_{5,0}$ 40 $v_{6,1} = v_{5,1} \wedge v_{5,2}$ 41 $v_{6,2} = v_{5,2}$ 42 $v_{2,6,3} = v_{5,3}$ 43 $v_{2,7,0} = v_{6,1}$ 44 $v_{7,1} = v_{6,0}$ 45 $v_{7,2} = v_{6,2}$ 46 $v_{7,3} = v_{7,3}$ 47 $v_{0,0} = v_{7,0}$ 48 $v_{0,1} = v_{7,1}$ 49 $v_{0,2} = v_{7,2}$ 50 $v_{0,3} = v_{7,3}$ 51 # Output 52 # function 53 $OUT[0] = v_{7,2}$ 54 # test implementation 57 IN = [0x0, 0x0] 59 SIMOM32_PERM(IN, OUT) 60 print('OUT', str([hex(i) for i in IN]))) 61 print('OUT', str([hex(i) for i in OUT]))	$v_{2} = v_{1} = v_{1$
29 $\sqrt{3}2 = \sqrt{2}2 \ \& \ v_2.3$ 30 $v_3.3 = v_2.3$ 31 $v_4.0 = v_3.0$ 32 $v_4.1 = v_3.1 \land v_3.2$ 33 $v_4.2 = v_3.2$ 34 $v_4.3 = v_3.3$ 35 $v_5.0 = v_4.0$ 36 $v_5.1 = v_4.1$ 37 $v_5.2 = ROT((v_4.0, 2, 16))$ 38 $v_5.3 = v_4.3$ 39 $v_6.0 = v_5.0$ 40 $v_6.0 = v_5.0$ 40 $v_6.1 = v_5.1 \land v_5.2$ 41 $v_6.2 = v_5.2$ 42 $v_6.3 = v_5.3$ 43 $v_7.0 = v_6.1$ 44 $v_7.1 = v_6.0$ 45 $v_7.2 = v_6.2$ 46 $v_7.2 = v_6.2$ 47 $v_0.0 = v_7.0$ 48 $v_0.1 = v_7.1$ 49 $v_0.2 = v_7.2$ 50 $v_0.3 = v_7.3$ 51 $\#$ Output 52 $\#$ Output 53 OUT[0] = v_7.0 54 $v_7 = [0.0, 0.00]$ 55 $III = [0.0, 0.00]$ 56 $\#$ test implementation 57 $II = [0.0, 0.00]$ 59 $SIMON32 PERM(IN, OUT)$ 60 $print(`UV', str([hex(i) for i in UN]))$ 61 $print(`UV', str([hex(i) for i in OUT]))$	
30 $\sqrt{-3} = \sqrt{-2}$ 31 $\sqrt{-4} = \sqrt{-3}$ 32 $\sqrt{-4} = \sqrt{-3}$ 33 $\sqrt{-4} = \sqrt{-3}$ 34 $\sqrt{-4} = \sqrt{-3}$ 35 $\sqrt{-4} = \sqrt{-3}$ 36 $\sqrt{-5} = \sqrt{-4}$ 37 $\sqrt{-5} = 80TL(v_4, \theta, 2, 16)$ 38 $\sqrt{-5} = \sqrt{-4}$ 39 $\sqrt{-6} = \sqrt{-5}$ 40 $\sqrt{-6} = \sqrt{-5}$ 40 $\sqrt{-6} = \sqrt{-5}$ 41 $\sqrt{-6} = \sqrt{-5}$ 42 $\sqrt{-6} = \sqrt{-5}$ 43 $\sqrt{-7} = \sqrt{-6}$ 44 $\sqrt{-7} = \sqrt{-6}$ 45 $\sqrt{-7} = \sqrt{-6}$ 46 $\sqrt{-7} = \sqrt{-6}$ 47 $\sqrt{-9} = \sqrt{-6}$ 48 $\sqrt{-2} = \sqrt{-7}$ 49 $\sqrt{-6} = \sqrt{-7}$ 49 $\sqrt{-6} = \sqrt{-7}$ 40 $\sqrt{-6} = \sqrt{-7}$ 40 $\sqrt{-7} = \sqrt{-6}$ 41 $\sqrt{-7} = \sqrt{-6}$ 42 $\sqrt{-3} = \sqrt{-7}$ 43 $\sqrt{-7} = \sqrt{-7}$ 44 $\sqrt{-7} = \sqrt{-7}$ 55 $\#$ test implementation 57 IN = [0x0, 0x0] 59 SIMON32_PERM(IN, OUT) 60 print('OUT', str([hex(1) for i in IN]))) 61 print('OUT', str([hex(1) for i in OUT]))	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
32 $\sqrt{a^{-1}} = \sqrt{3} (1 + \sqrt{3})^{2}$ 33 $\sqrt{a^{-1}} = \sqrt{3} (1 + \sqrt{3})^{2}$ 34 $\sqrt{4} (2 + \sqrt{3})^{2}$ 35 $\sqrt{5} (0 + \sqrt{3})^{2}$ 36 $\sqrt{5} (0 + \sqrt{3})^{2}$ 37 $\sqrt{5} (2 + \sqrt{3})^{2}$ 38 $\sqrt{5} (2 + \sqrt{3})^{2}$ 39 $\sqrt{6} (2 + \sqrt{5})^{2}$ 40 $\sqrt{5} (2 + \sqrt{5})^{2}$ 41 $\sqrt{5} (2 + \sqrt{5})^{2}$ 42 $\sqrt{6} (2 + \sqrt{5})^{2}$ 43 $\sqrt{7} (2 + \sqrt{5})^{2}$ 44 $\sqrt{7} (2 + \sqrt{5})^{2}$ 45 $\sqrt{7} (2 + \sqrt{5})^{2}$ 46 $\sqrt{7} (3 + \sqrt{5})^{2}$ 47 $\sqrt{6} (2 + \sqrt{5})^{2}$ 48 $\sqrt{7} (2 + \sqrt{5})^{2}$ 49 $\sqrt{2} (2 + \sqrt{5})^{2}$ 49 $\sqrt{2} (2 + \sqrt{7})^{2}$ 50 $\sqrt{2} (2 + \sqrt{7})^{2}$ 51 $\sqrt{2} (2 + \sqrt{7})^{2}$ 52 $\frac{\#}{2} (2 + \sqrt{7})^{2}$ 53 $0 0 0 T [0] = \sqrt{7} (2 + \sqrt{7})^{2}$ 54 $\frac{1}{2} (2 + \sqrt{7})^{2}$ 55 $\frac{1}{2} \frac{1}{2} (2 + \sqrt{7})^{2}$ 56 $\frac{1}{2} (2 + \sqrt{7})^{2}$ 57 $\frac{1}{2} \frac{1}{2} (2 + \sqrt{7})^{2}$ 58 $\frac{1}{2} \frac{1}{2} \frac{1}{2$	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	$v_4_0 = v_3_0$
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	$v_4_1 = v_3_1 \wedge v_3_2$
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	v 5 0 = v 4 0
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	x = 51 = x = 41
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
45 v,7.2 = v.6.2 46 v,7.3 = v.6.3 47 v.0.0 = v.7.0 48 v.0.1 = v.7.1 49 v.0.2 = v.7.2 50 v.0.3 = v.7.3 51 # Output 52 # Output 53 OUT[0] = v.7.0 54 OUT[1] = v.7.1 55 # test implementation 56 # test implementation 57 IN = [0x0, 0x0] 59 SIMON32_PERM(IN, OUT) 60 print('IW', str([hex(i) for i in IN])) 61 print('UV', str([hex(i) for i in OUT]))	$v_{0} = v_{6}$
46 v_7_3 = v_6_3 47 v_0 e v.7_0 48 v_0_1 = v.7_1 49 v_0_2 = v.7_2 50 v_0_3 = v.7_3 51 # Output 53 OUT[0] = v.7_0 54 OUT[1] = v.7_0 55 # test implementation 57 IN = [0x0, 0x0] 59 SIMON32 PERM(IN, OUT) 60 print('UT', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))	
47 v.0.9 = v.7.0 48 v.0.1 = v.7.1 49 v.0.2 = v.7.2 50 v.0.3 = v.7.3 51 # Output 53 OUT[0] = v.7.0 54 • 0.1[1] = v.7.1 55 # test implementation 56 # test implementation 57 IN = [0x0, 0x0] 59 SIMON32 PERM(IN, OUT) 60 print('IW', str([hex(i) for i in IN])) 61 print('UW', str([hex(i) for i in OUT])))	
47 v.0.0 = v.7.0 48 v.0.1 = v.7.1 49 v.0.2 = v.7.2 50 v.0.3 = v.7.3 51 # Output 53 OUT[0] = v.7.0 54 • 0.1[1] = v.7.1 55 # test implementation 56 # test implementation 57 IN = [0x0, 0x0] 59 SIMON32 PERM(IN, OUT) 60 print('IW', str([hex(i) for i in IN])) 61 print('UW', str([hex(i) for i in OUT])))	$v_7_3 = v_6_3$
48 v.0.1 = v.7.1 49 v.0.2 = v.7.2 50 v.0.3 = v.7.3 51 # Output 53 OUT[0] = v.7.0 54 OUT[1] = v.7.1 55 # test implementation 57 IN = [0x0, 0x0] 59 SIMON32_PERN(IN, 0UT) 60 print('IN', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))	
49 v_02 = v_72 50 v_03 = v_73 51 # Output 52 # Output 53 OUT[0] = v_7_0 54 OUT[1] = v_7_1 55 # test implementation 57 IN = [0x0, 0x0] 58 OUT = [0x0, 0x0] 59 SIMON32 PERM(IN, OUT) 60 print('IW', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT])))	
50 v_0_3 = v_7_3 51 # Output 52 # Output 53 OUT[0] = v_7_0 54 OUT[1] = v_7_1 55 56 # test implementation 57 IN = [0x0, 0x0] 58 OUT = [0x0, 0x0] 59 SIMON32_PERM(IN, OUT) 60 print('IN', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))	
<pre>51 # Output 52 # Output 53 OUT[0] = v_7_0 54 OUT[1] = v_7_0 55 # test implementation 57 IN = [0x0, 0x0] 58 OUT = [0x0, 0x0] 59 SIMON32_PERN(IN, OUT) 60 print('IN', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))</pre>	
<pre>52 # Output 53 OUT[0] = v.Z_0 54 OUT[1] = v.Z_1 55 # test implementation 57 IN = [0x0, 0x0] 58 OUT = [0x0, 0x0] 59 SIMON32_PERM(IN, OUT) 60 print('IN', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))</pre>	*_ <u>_</u>
53 OUT[0] = v_7_0 54 OUT[1] = v_7_1 55 56 # test implementation 57 IN = [0x0, 0x0] 58 OUT = [0x0, 0x0] 59 SIMON32_PERN(IN, OUT) 60 print('IN', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))	
<pre>54</pre>	
<pre>55 # test implementation 57 IN = [0x0, 0x0] 58 OUT = [0x0, 0x0] 59 SIMON32_PERM(IN, OUT) 60 print('IN', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))</pre>	
 56 # test implementation 57 IN = [0x0, 0x0] 58 OUT = [0x0, 0x0] 59 SIMON32_PERN(IN, OUT) 60 print('IN', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT])) 	$001[1] = v_/_1$
<pre>57 IN = [0x0, 0x0] 58 OUT = [0x0, 0x0] 59 SIMON32_PERM(IN, OUT) 60 print('IN', Str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))</pre>	
58 OUT = [0x0, 0x0] 59 SIMON32_PERM(IN, OUT) 60 print('IN', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))	
<pre>58 OUT = [0x0, 0x0] 59 SIMON32_PERM(IN, OUT) 60 print('IN', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))</pre>	IN = [0x0, 0x0]
<pre>60 print('IN', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))</pre>	$OUT = [0 \times 0, 0 \times 0]$
<pre>60 print('IN', str([hex(i) for i in IN])) 61 print('OUT', str([hex(i) for i in OUT]))</pre>	
<pre>61 print('OUT', str([hex(i) for i in OUT]))</pre>	
	<pre>print('OUT', str([hex(i) for i in OUT]))</pre>

#include <stdint.h> #include <stdio.h>

	//Rotation Macros
	<pre>#define ROTL(n, d, bitsize) (((n << d) (n >> (bitsize - d))) & ((1<<bitsize) -="" 1))<="" pre=""></bitsize)></pre>
	#define ROTR(n, d, bitsize) ((($n > d$) ($n < (bitsize - d)$)) & ((1 (bitsize) - 1))
	α
	// Function implementing the SIMON32_PERM function
	// Input:
	// IN: an array of 2 words of 16 bits
11	// Output:
12	// OUT: an array of 2 words of 16 bits
13	void SIMON32_PERM(uint32_t* IN, uint32_t* OUT){
14	uint32_t v_0_0, v_0_1, v_0_2, v_0_3, v_1_0, v_1_1, v_1_2, v_1_3, v_2_0, v_2_1, v_2_2, v_2_3,
	// Input
	$v_0_0 = IN[0];$
	$v_0 - 1 = IN[1];$
20	// Round function
	for (int i=0; i<10; i++) {
	v_1_0 = v_0_0;
	v_1_1 = v_0_1;
	$v_{1_2} = ROTL(v_{0_0}, 1, 16);$
25	v_1_3 = v_0_3;
	$v_{-2-0} = v_{-1-0}$;
	$v_2 1 = v_1 1;$
	$v_{2,2} = v_{1,2}$
	v_2_3 = ROTL(v_1_0, 8, 16);
	$v_3_0 = v_2_0;$
	v_3_1 = v_2_1;
	v_3_2 = v_2_2 & v_2_3;
	$v_{3}^{3} = v_{2}^{2};$
	$v_{-4}^{-0} = v_{-3}^{-3} 0;$
	v_4_1 = v_3_1 ^ v_3_2;
36	v_4_2 = v_3_2;
	v_4_3 = v_3_3;
	v_5_0 = v_4_0;
	v_5_1 = v_4_1;
	v_5_2 = ROTL(v_4_0, 2, 16);
	$v_{5}^{-3} = v_{4}^{-3};$
42	v = -5
43	v_6_1 = v_5_1 ^ v_5_2;
44	
	v_6_2 = v_5_2;
	v_6_3 = v_5_3;
	$v_7_0 = v_6_1;$
47	v_7_1 = v_6_0;
	v_7_2 = v_6_2;
	$v_{-7-3} = v_{-6-3};$
	v - 0 = v - 7 = 0;
	$v_0 - 1 = v_0 - 7 - 1;$
	v_0_2 = v_7_2;
53	v_0_3 = v_7_3;
	}
	// Output
	$OUT[0] = v_7_0;$
	$\operatorname{out}[1] = \sqrt{-7} \cdot 1;$
	}

C (rolled/unrolled)



Python (rolled/unrolled)

Automatic Generation of SAT / MILP models

-1802 1818 -1834 -1803 -1819 -1835 0 -1802 -1818 1834 -1803 -1819 -1835 0 1803 1819 -1835 1804 1820 1836 0 1803 -1819 1835 1804 1820 1836 0 -1803 1819 1835 1804 1820 1836 0 -1803 -1819 -1835 1804 1820 1836 0 1803 1819 1835 -1804 -1820 -1836 0 1803 -1819 -1835 -1804 -1820 -1836 0 -1803 1819 -1835 -1804 -1820 -1836 0 -1803 -1819 1835 -1804 -1820 -1836 0 1804 1820 -1836 1805 1821 1837 0 1804 -1820 1836 1805 1821 1837 0 -1804 1820 1836 1805 1821 1837 0 -1804 -1820 -1836 1805 1821 1837 0 1804 1820 1836 -1805 -1821 -1837 0 1804 -1820 -1836 -1805 -1821 -1837 0 -1804 1820 -1836 -1805 -1821 -1837 0 -1804 -1820 1836 -1805 -1821 -1837 0 1805 1821 -1837 1806 1822 1838 0 1805 -1821 1837 1806 1822 1838 0 -1805 1821 1837 1806 1822 1838 0 -1805 -1821 -1837 1806 1822 1838 0 1805 1821 1837 -1806 -1822 -1838 0 1805 -1821 -1837 -1806 -1822 -1838 0 -1805 1821 -1837 -1806 -1822 -1838 0 -1805 -1821 1837 -1806 -1822 -1838 0 1806 1822 -1838 1807 1823 1839 0 1806 -1822 1838 1807 1823 1839 0 -1806 1822 1838 1807 1823 1839 0 -1806 -1822 -1838 1807 1823 1839 0 1806 1822 1838 -1807 -1823 -1839 0 1806 -1822 -1838 -1807 -1823 -1839 0

vs_3_1_0_10 - vs_3_1_1_10 + ADD_3_1_0_p_9 >= 0 vs_3_2_0_10 - vs_3_1_0_10 + ADD_3_1_0_p_9 >= 0 vs_3_1_0_10 + vs_3_1_1_10 + vs_3_2_0_10 + ADD_3_1_0_p_9 <= 3 vs_3_1_0_10 + vs_3_1_1_10 + vs_3_2_0_10 - ADD_3_1_0_p_9 >= 0 vs_3_1_0_9 + vs_3_1_1_9 + vs_3_2_0_9 + ADD_3_1_0_p_9 - vs_3_1_1_10 >= 0 vs_3_1_1_10 + vs_3_1_0_9 - vs_3_1_1_9 + vs_3_2_0_9 + ADD_3_1_0_p_9 >= 0 $vs_3_1_1_0 - vs_3_1_0_9 + vs_3_1_1_9 + vs_3_2_0_9 + ADD_3_1_0_p_9 >= 0$ $vs_3_1_0_10 + vs_3_1_0_9 + vs_3_1_1_9 - vs_3_2_0_9 + ADD_3_1_0_p_9 \ge 0$ vs_3_2_0_10 - vs_3_1_0_9 - vs_3_1_1_9 - vs_3_2_0_9 + ADD_3_1_0_p_9 >= -2 vs_3_1_0_9 - vs_3_1_1_10 - vs_3_1_1_9 - vs_3_2_0_9 + ADD_3_1_0_p_9 >= -2 vs_3_1_1_9 - vs_3_1_1_10 - vs_3_1_0_9 - vs_3_2_0_9 + ADD_3_1_0_p_9 >= -2 vs 3 2 0 9 - vs 3 1 1 10 - vs 3 1 0 9 - vs 3 1 1 9 + ADD 3 1 0 p 9 >= -2 vs 3 1 1 11 - vs 3 2 0 11 + ADD 3 1 0 p 10 >= 0 vs_3_1_0_11 - vs_3_1_1_11 + ADD_3_1_0_p_10 >= 0 vs_3_2_0_11 - vs_3_1_0_11 + ADD_3_1_0_p_10 >= 0 vs_3_1_0_11 + vs_3_1_1_11 + vs_3_2_0_11 + ADD_3_1_0_p_10 <= 3 vs_3_1_0_11 + vs_3_1_1_11 + vs_3_2_0_11 - ADD_3_1_0_p_10 >= 0 $vs_3_1_0_10 + vs_3_1_1_10 + vs_3_2_0_10 + ADD_3_1_0_p_10 - vs_3_1_1_11 >= 0$ vs_3_1_1_11 + vs_3_1_0_10 - vs_3_1_1_10 + vs_3_2_0_10 + ADD_3_1_0_p_10 >= 0 vs_3_1_1_11 - vs_3_1_0_10 + vs_3_1_1_10 + vs_3_2_0_10 + ADD_3_1_0_p_10 >= 0 vs_3_1_0_11 + vs_3_1_0_10 + vs_3_1_1_10 - vs_3_2_0_10 + ADD_3_1_0_p_10 >= 0 vs 3 2 0 11 - vs 3 1 0 10 - vs 3 1 1 10 - vs 3 2 0 10 + ADD 3 1 0 p 10 >= -2 vs_3_1_0_10 - vs_3_1_1_11 - vs_3_1_1_10 - vs_3_2_0_10 + ADD_3_1_0_p_10 >= -2 $vs_3_1_1_0 - vs_3_1_1_11 - vs_3_1_0_10 - vs_3_2_0_10 + ADD_3_1_0_p_10 >= -2$ vs_3_2_0_10 - vs_3_1_1_11 - vs_3_1_0_10 - vs_3_1_1_10 + ADD_3_1_0_p_10 >= -2 vs_3_1_1_12 - vs_3_2_0_12 + ADD_3_1_0_p_11 >= 0 vs_3_1_0_12 - vs_3_1_1_12 + ADD_3_1_0_p_11 >= 0 vs_3_2_0_12 - vs_3_1_0_12 + ADD_3_1_0_p_11 >= 0 vs 3 1 0 12 + vs 3 1 1 12 + vs 3 2 0 12 + ADD 3 1 0 p 11 <= 3 $vs_3_1_0_12 + vs_3_1_1_12 + vs_3_2_0_12 - ADD_3_1_0_p_11 >= 0$ vs_3_1_0_11 + vs_3_1_1_11 + vs_3_2_0_11 + ADD_3_1_0_p_11 - vs_3_1_1_12 >= 0 vs_3_1_1_12 + vs_3_1_0_11 - vs_3_1_1_11 + vs_3_2_0_11 + ADD_3_1_0_p_11 >= 0 vs_3_1_1_12 - vs_3_1_0_11 + vs_3_1_1_11 + vs_3_2_0_11 + ADD_3_1_0_p_11 >= 0 vs 3 1 0 12 + vs 3 1 0 11 + vs 3 1 1 11 - vs 3 2 0 11 + ADD 3 1 0 p 11 >= 0 vs 3 2 0 12 - vs 3 1 0 11 - vs 3 1 1 11 - vs 3 2 0 11 + ADD 3 1 0 p 11 >= -2 vs_3_1_0_11 - vs_3_1_1_12 - vs_3_1_1_11 - vs_3_2_0_11 + ADD_3_1_0_p_11 >= -2 vs_3_1_1_11 - vs_3_1_1_12 - vs_3_1_0_11 - vs_3_2_0_11 + ADD_3_1_0_p_11 >= -2 vs_3_2_0_11 - vs_3_1_1_12 - vs_3_1_0_11 - vs_3_1_1_11 + ADD_3_1_0_p_11 >= -2 vs_3_1_1_13 - vs_3_2_0_13 + ADD_3_1_0_p_12 >= 0

MILP

SAT



NANYANG TECHNOLOGICAL UNIVERSITY | SINGAPORE

One operator – several modelisations

Current Operators: Equal, NOT, AND, OR, XOR, NXOR, Sbox, Modular Addition, Rotation/Shift, Matrix multiplication, Constant Addition

Soon to be added: AES round, Modular multiplication

One operator can have several types of modelisation !

Ex1: with an 8-bit variable, we can track exact or truncated differences Ex2: Modular Addition can have different strategies of modeling

In OCP, you can switch any operator from one type of modelisation to another very easily (simply change a flag).

=> You can have entire modeling strategies where some operators behave differently than others (Ex: SHA attacks with linear/non linear)



Ciphers already implemented

- AES
- Ascon
- GIFT
- ROCCA
- Simon
- SipHash
- Skinny
- Speck
- ChaCha (probably this week ⁽ⁱ⁾)
- more to come soon ...

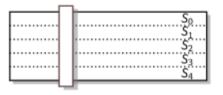


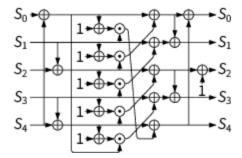
One primitive – several modelisations

One cipher can have several modelisations !

Ex1: ASCON state with 5 64-bit variables or 320 1-bit variables Ex2: SuperSbox representation for a few rounds

In OCP, you can choose one type of modelisation very easily (simply set a flag).





So
S1

 $S_0 := S_0 \bigoplus (S_0 \Longrightarrow 19) \bigoplus (S_0 \ggg 28)$ $S_1 := S_1 \bigoplus (S_1 \ggg 61) \bigoplus (S_1 \ggg 39)$ $S_2 := S_2 \bigoplus (S_2 \ggg 1) \bigoplus (S_2 \ggg 6)$ $S_3 := S_3 \bigoplus (S_3 \ggg 10) \bigoplus (S_3 \ggg 17)$ $S_4 := S_4 \bigoplus (S_4 \ggg 7) \bigoplus (S_4 \ggg 41)$

ASCON round



OCP Current State

Current capabilities of OCP:

- Differential cryptanalysis (single or related key)
- Truncated differential cryptanalysis (single or related key)
- Implementations automatically output test vectors, makes sure your entire model is correct

Coming soon: linear cryptanalysis, differential-linear cryptanalysis, additional modeling improvements (Matsui branch and bound option), additional modeling of the operators (window heuristic for mod addition, etc.)

Help us include more attacks / models !

Issue: inverse of primitives ?



Documentation

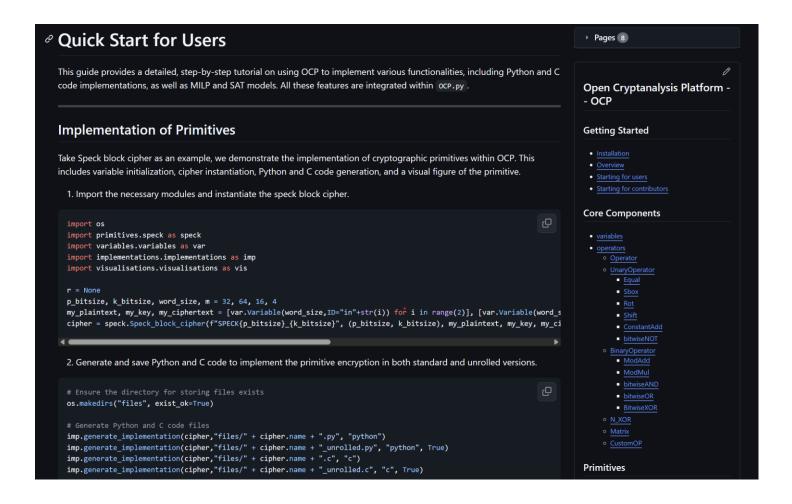
https://github.com/Open-CP/OCP/wiki

'Operator' Class	Pages 7
The "Operator class" is a fundamental component in our graph modeling framework, which represents a constraint/operator object. It represents a type of node that can only be linked to Variable nodes, handling operations or constraints between groups of variables.	Open Cryptanalysis Platform - - OCP
Attributes	Getting Started • Installation
 input_vars : List of input variables associated with the operator. 	Overview Starting for users
 output_vars : List of output variables associated with the operator. 	Starting for contributors
 model_version : Integer indicating the version of the model that this operator is associated with. 	
• ID : Unique identifier string for the operator.	Core Components
	variables
	operators
Methods	• Operator
	• <u>UnaryOperator</u>
	Equal Sbox
display()	= <u>350x</u> = Rot
Returns the name of the operator's class.	■ <u>Shift</u>
	 ConstantAdd
Outputs:	bitwiseNOT
	BinaryOperator ModAdd
 Returns a string representing the name of the operator's class. 	 ModMul
	bitwiseAND
Example:	bitwiseOR



Documentation (Quick Start for Users)

https://github.com/Open-CP/OCP/wiki





Documentation (Quick Start for Contributors)

https://github.com/Open-CP/OCP/wiki

Quick Start for Contributors	Pages 9
This guide provides instructions for contributing to OCP.	<i>P</i> Open Cryptanalysis Platform -
Contribution to Operators	- OCP
1. Adding New Operators	Getting Started • Installation
 Define a new operator class in operators/operators.py . Implement the operator's functionality and modeling methods in alignment with existing operators. 	<u>Overview</u> <u>Starting for users</u> <u>Starting for contributors</u>
2. Extending Existing Operators	Core Components
 Add new Python and C implementations to optimize performance in generate_implementation(). Add new modeling techniques (e.g., "MILP", "SAT") by specifying model_type in generate_model() 	• <u>variables</u> • <u>operators</u>
 Add new modeling versions (e.g., "diff", "truncated_diff", "linear") for various cryptanalysis techniques by specifying model_version in generate_model(). 	 Operator UnaryOperator Equal Sbox
Contribution to Primitives	= <u>Stox</u> = <u>Rot</u> = <u>Shift</u> = ConstantAdd
1. Adding New Primitives	bitwiseNOT o BinaryOperator ModAdd
• Create a new_primitive.py file in primitives/.	 ModMul



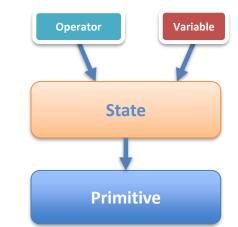
Main TODOs: Core and Attack Module

• Core OCP:

- Growing the pre-existing library of ciphers
- Growing the pre-existing library of operators
- Growing the number of different models/representations for each operators/ciphers
- Conversions between variable types ?
- Operating modes ?

Attacks module:

- More attack types ! (linear / differential-linear / boomerang / impossible diff / division property / etc.)
- Key recovery phase (started incorporating Autoguess)
- Pre-existing library of attacks
- Standardized benchmarks for comparing attacks
- Allow modular combination of attacks/models
- not limited to MILP and SAT ! Ad-hoc algorithms also welcome



Attacks module



Main TODOs: Other Modules

Implementations module:

- **VHDL**, Rust, other use cases?
- Automatic verification of test vectors for new representations
- Faster C implementations
- Optimized Sbox / Diffusion matrix implementations database
- Long-term future: side-channels resistant implementations

Solving module:

- CP? ML? Others?
- More MILP and SAT solvers
- Parallelization

Visualisation module:

- Automated visualisations of the attacks
- Graphical interface for user interaction (cipher design / attack config.)
- Automated generation of LaTeX/TikZ figures !!!

Implementations module

> Solving module

GUI / Visualisation module



OCP - Organisation

We need to establish (simple) **governance** to have proper:

- development processes into place,
- responsibilities,
- communication, regular meetings, valuable feedback.

One person responsible for each (sub)module ?

Philosophy:

- Remain Free and Open Source (careful if you copy code!)
- Don't rely on external tools unless really necessary (keep install simple)
- Keep code clean and compact
- Keep code generic !



We want YOU !

WE WANT YOU!

If interested to contribute / getting updates:

- contact <u>thomas.peyrin@ntu.edu.sg</u>
- or join the googlegroup
- automated-cryptanalysis@googlegroups.com
- or click on this link:

https://groups.google.com/g/automated-cryptanalysis

• GitHub:

https://github.com/Open-CP/OCP

Thanks to all the cryptanalysts that already joined the list and gave feedback !





Thank You !

Grazie!



NANYANG TECHNOLOGICAL UNIVERSITY | SINGAPORE