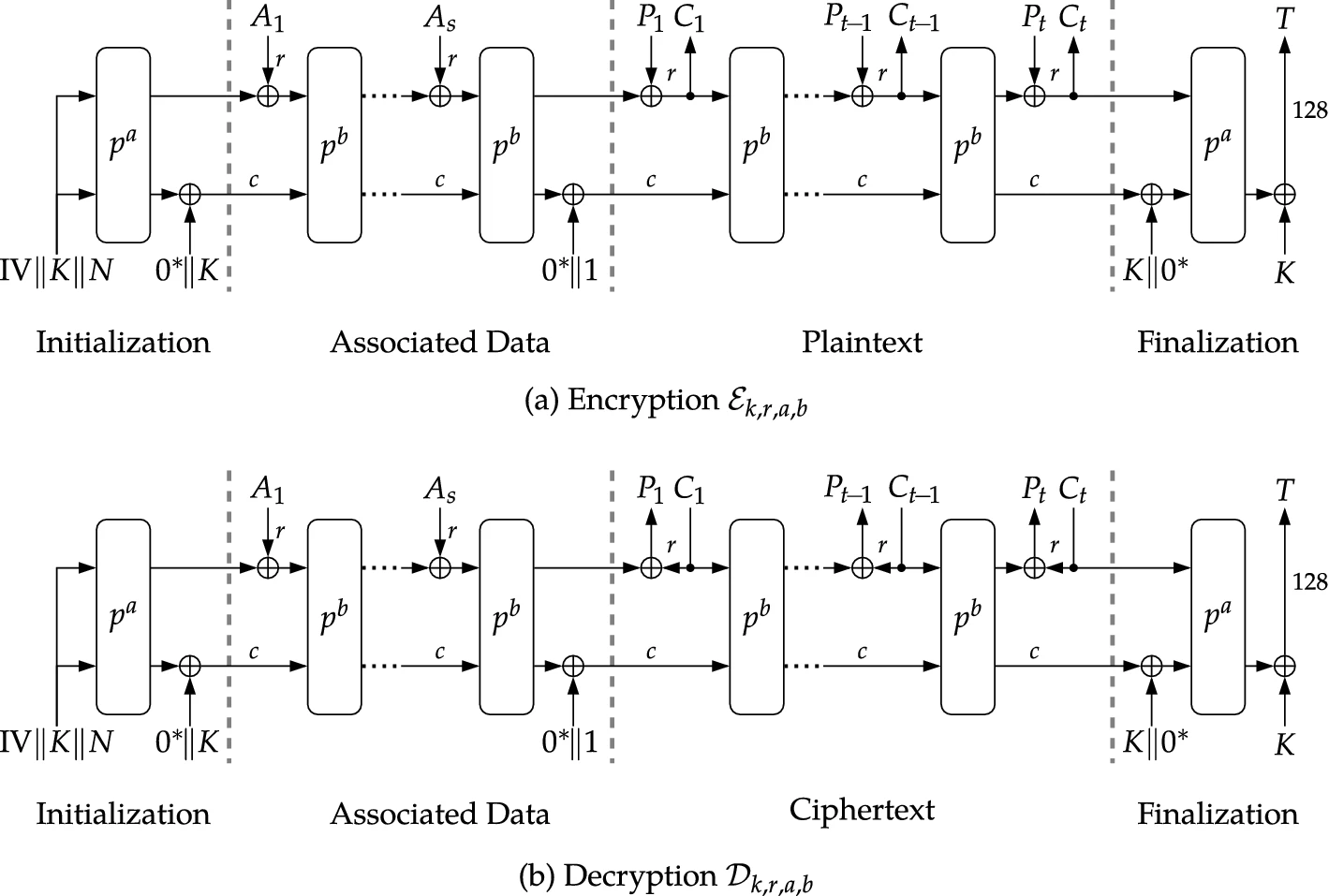
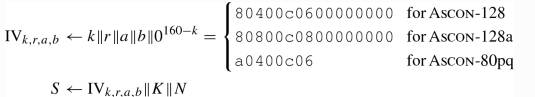
**Monkey Duplex Sponge**

****

Initialization The 320-bit initial state of ASCON is formed by the secret key K of k bits and nonce N of 128 bits, as well as an IV specifying the algorithm (including the key size k, the rate r, the initialization and finalization round number a, and the intermediate round number b, each written as an 8-bit integer):



In the initialization, a rounds of the transformation p are applied to the initial state, followed by an XOR of the secret key K:

Processing Associated Data ASCON processes the associated data A in blocks of r bits. It appends a single 1 and the smallest number of 0s to A to obtain a multiple of r bits and split it into s blocks of r bits, A1 ∥ … ∥ As A1 ‖ … ‖ As. In case A is empty, no padding is applied and s=0:

Processing Plaintext/Cipher text ASCON processes the plaintext P in blocks of r bits. The padding process appends a single 1 and the smallest number of 0s to the plaintext P such that the length of the padded plaintext is a multiple of r bits. The resulting padded plaintext is split into t blocks of r bits,

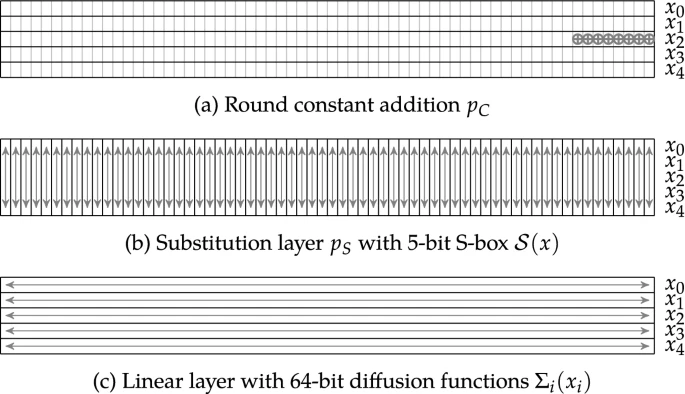
**Permutation**

The main components of the schemes ASCON, ASCON-XOF, and ASCON-HASH are the two 320-bit permutations  and. The permutations iteratively apply an SPN-based round transformation p that in turn consists of three steps

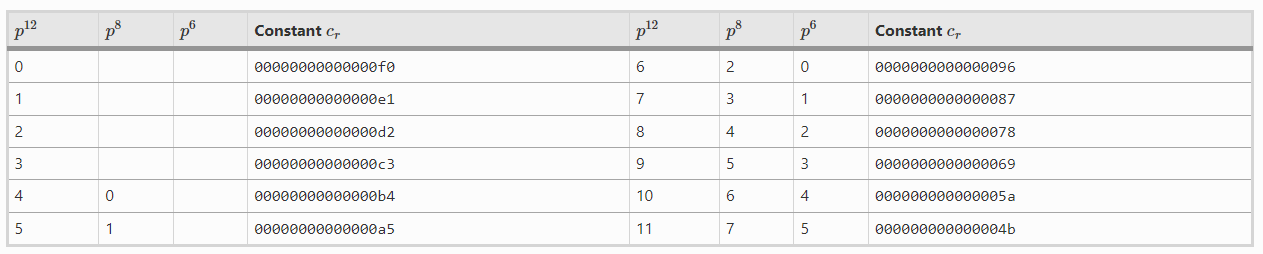
p = ∘  ∘  .

 and   differ only in the number of rounds. The number of rounds a and the number of rounds b are tunable security parameters.

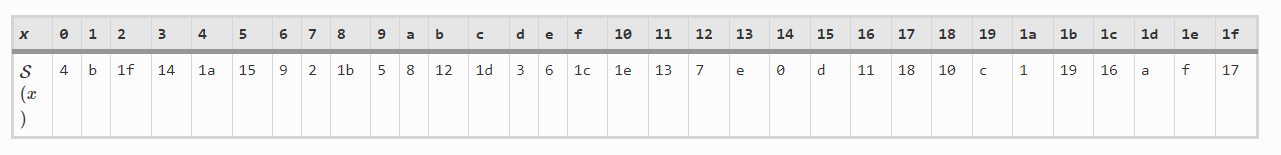
For the description and application of the round transformations, the 320-bit state S is split into five 64-bit registers words xi, S =



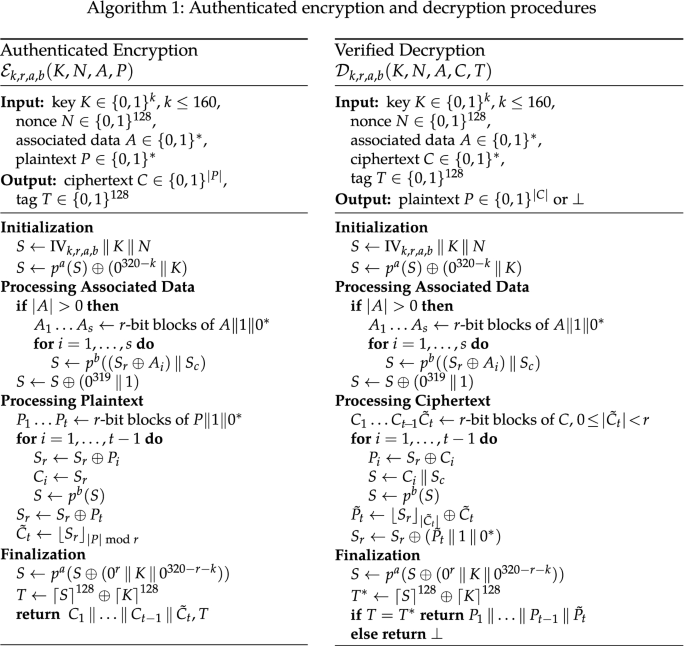
Addition of Constants the constant addition step  adds a round constant cr to register word  of the state S in round I. Both indices r and i start from zero and we use r=i for  and r=i+a−b for



Substitution Layer The substitution layer pS updates the state S with 64 parallel applications of the 5-bit S-box S(x) to each bit-slice of the five registers …  . It is typically implemented in this bit sliced form with operations performed on the entire 64-bit words, The lookup table of SS is given where  is the MSB and x4 the LSB.



**Pseudocode**

****

Let 𝑊 = FSW [𝑝, 𝑟, 𝑘, 𝑛, ] be an instance of FSW. Denote any query to 𝑊.initialize and a list of subsequent queries to 𝑊.wrap by (𝑁,(𝐴1, 𝑀1), . . . ,(𝐴𝑛, 𝑀𝑛)). Then, FSW injectively maps this sequence to a sequence of corresponding FKD duplexing queries (𝑄1, . . . , 𝑄𝑑). Proof. We prove the injectivity of the mapping by showing how it can be inverted. We refer to the mapping 𝑄 of (3) to argue that every 𝑄𝑖 can be split into three strings 𝐿 , 𝐹𝑖 , 𝑅𝑖 with |𝐿𝑖 | = 𝑟 + 1, |𝐹𝑖 | = 3 and |𝑅𝑖 | ≤ 𝑐 − 5 . The main trick is to use the frame bits used in FSW to determine boundaries of wrapping queries and their logical parts. We will refer to the FKD queries as “frames”. We can recover the AD-message pairs (in the following just “pair”) from Q = (𝑄1, . . . , 𝑄𝑑) in a left-to-right fashion. Any pair (𝐴, 𝑀) is encoded in a subsequence of Q that starts by a frame with frame bits 𝐹N and ends by a frame just before the next frame with frame bits 𝐹N. Depending on the lengths of 𝐴 and 𝑀, the pattern of frame bits between these boundary frames can differ as depicted in Fig. 6. If both 𝐴 and 𝑀 are non-empty, we follow the edge marked as A. If there is the same number of 𝑟-bit blocks in 𝑀 as there is of 𝑐 − 5 bit blocks in 𝐴, then we follow the path A.1. Otherwise we follow the path A.2 and then A.21 if there were fewer blocks in 𝐴 than in 𝑀 and the path A.22 if there were in turn more blocks in 𝐴 than in 𝑀. If 𝑀 ̸= 𝐴 = 𝜀, then we follow the path B; if 𝐴 ̸= 𝑀 = 𝜀 we follow the path C. In a special case, where both 𝐴 = 𝑀 = 𝜀, we follow path D. We can see, that every possible case of lengths of 𝑀 and 𝐴 in terms of blocks yields a distinct pattern of frame bit sequences. Having identified which path in Fig. 6 we are following, we can recover 𝐴 and 𝑀. Every frame 𝑄𝑖 with 𝐹𝑖 ∈ {𝐹AM, 𝐹AM|} holds a padded block of 𝑀 in 𝐿𝑖 and an unpadded block of 𝐴 in 𝑅 . If 𝐹𝑖 = 𝐹M, then there is a padded block of 𝑀 in 𝐿𝑖 and 𝑅𝑖 = 𝜀. If 𝐹𝑖 = 𝐹A, then there is a padded block of 𝐴 in 𝐿𝑖 and another unpadded block of 𝐴 in 𝑅 . The frames with 𝐹𝑖 ∈ {𝐹¯AM, 𝐹¯M, 𝐹¯A} are used to produce the tag and are thus treated specially. The first frame with 𝐹¯ 𝜒 holds data blocks and the following ones do not. If 𝜒 = AM, then there is a padded block of 𝑀 in 𝐿𝑖 and an unpadded block of 𝐴 in 𝑅 . If 𝜒 = M, then there is only a padded block of 𝑀 in 𝐿 . If 𝜒 = A and we are not on path D then there is a padded block of 𝐴 in 𝐿𝑖 and a following unpadded block of 𝐴 in 𝑅𝑖 . If we are on path D then none of the frames holds any data, since both 𝐴 and 𝑀 are empty