

# ASCON v1.2

## Analisi e implementazione

CdL in Sicurezza dei Sistemi e delle Reti Informatiche

Melissa Moioli, Mattia Perfumo, Tiziano Radicchi  
09831A, xxx, 12172A

May 6, 2024

### Abstract

*Si propone un'analisi della famiglia di schemi di cifratura autenticata e hashing ASCON, selezionata come soluzione primaria per la realizzazione di lightweight cryptography nella competizione CAESAR nonché come famiglia adatta a standardizzazione NIST. Si propone in aggiunta una implementazione di ASCON-128, nonché una dimostrazione di utilizzo del cifrario in sistemi embedded tramite un semplice protocollo di comunicazione.*



# Contents

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Agenda . . . . .	1
<b>2</b>	<b>Specifica</b>	<b>2</b>
2.1	Introduzione . . . . .	2
2.1.1	Authenticated Encryption . . . . .	2
2.1.2	Parametri . . . . .	2
2.1.3	State . . . . .	3
2.2	Authenticated Encryption . . . . .	3
2.3	Funzione p . . . . .	4
2.4	ASCON-128 . . . . .	5
<b>3</b>	<b>Design Rationale</b>	<b>6</b>
3.1	Sponge Function . . . . .	6
3.2	MonkeyDuplex . . . . .	6
<b>4</b>	<b>Crittoanalisi</b>	<b>7</b>
4.1	Security Claims . . . . .	7
4.2	Security Analysis . . . . .	7
4.2.1	Crittoanalisi differenziale e lineare . . . . .	7
<b>5</b>	<b>Protocollo di comunicazione</b>	<b>8</b>
5.1	Implementazione . . . . .	9

# 1 Introduzione

Con questo documento intendiamo proporre un'analisi e implementazione dello schema di cifratura **ASCON-128**, facente parte di una famiglia più ampia di cifrari e algoritmi di hashing. In particolare, sono presenti:

- *ASCON-128a*: Cifrario a blocchi di 128 bit anziché 64
- *ASCON-80pq*: maggiore robustezza contro ricerca della chiave tramite quantum-computing (chiave da 160 bit)
- *ASCON-HASHa*
- *ASCON-XOF*: Extendable output function per produrre hash outputs di lunghezza arbitraria
- *ASCON-XOFa*

Tutti gli schemi forniscono una sicurezza di 128-bit ed utilizzano una matrice di stato da 320-bit nei processi di cifratura, decifratura ed hashing. Le raccomandazioni NIST includono le combinazioni *ASCON-128 + ASCON-HASH* oppure *ASCON-128a + ASCON-HASHa*.

## 1.1 Agenda

Nelle sezioni successive si introdurrà in maniera più dettagliata il funzionamento ed i parametri per l'utilizzo del cifrario; si analizzeranno le strutture scelte, effettuando richiami e confronti (ove possibile) con gli argomenti affrontati durante l'insegnamento di *Crittografia*. Si proseguirà poi con una implementazione in linguaggio C del cifrario, con delle dimostrazioni di utilizzo, per concludere infine con una proposta di applicazione dello schema in uno scenario di comunicazione wireless tra sistemi embedded, al fine di ricreare uno scenario realistico e dimostrare l'utilizzabilità dello schema in ambienti con potenza di calcolo, memoria e energia limitate.

## 2 Specifica

### 2.1 Introduzione

La suite ASCON fornisce la cosiddetta **Authenticated Encryption with Associated Data (AEAD)**, nonché le funzionalità di hashing già citate.

Un cifrario autenticato consiste in uno schema di cifratura che possa assicurare, oltre alla più ovvia confidenzialità dei dati cifrati, l'integrità di questi, affinché il ricevente possa assicurarsi che il messaggio durante il transito non abbia subito distorsioni, dovute a disturbi o più importante a tampering da parte di un'entità terza. ASCON assicura l'integrità del messaggio trasmesso, nonché di eventuali dati associati. L'idea alla base è quella di restituire, al termine della cifratura, un **tag** da 128 bit estratto dallo state alla fine della procedura di cifratura. Lo state utilizzato per la cifratura, con riferimento ai diagrammi più avanti riportati, dipende anche da eventuali dati associati. In questo modo si può assicurare l'integrità anche di quei metadati per i quali non è necessaria la confidenzialità, ma per i quali ci si vuole comunque assicurare dell'assenza di manipolazioni da parte di threat actors.

Tutti gli schemi si basano su una funzione  $p$ , composta da somme, sostituzioni e permutazioni (più avanti approfondite) e che opera su una matrice di stato da 320 bit. La funzione è stata ideata per assicurare robustezza a fronte delle limitazioni hardware e software che un sistema embedded può imporre. Le operazioni sono **definite su parole da 64 bit** e trattandosi di **operazioni logiche** (AND, NOT, XOR, ROT), l'implementazione dello schema risulta particolarmente ottimale per ambienti di questo tipo. Inoltre, si adatta particolarmente a scenari in cui dispositivi trasmettono poche informazioni in maniera periodica: dai benchmark risulta che lo schema è particolarmente efficiente per messaggi brevi.

Non vi è necessità di operazioni inverse, dal momento in cui  $p^a$  e  $p^b$  sono valutati in una sola direzione sia per cifratura che decifratura.

ASCON non utilizza key scheduling o espansione di chiave. Non ci sono costi aggiuntivi quando la chiave viene cambiata.

#### 2.1.1 Authenticated Encryption

Per realizzare la cifratura con autenticazione dei dati gli algoritmi in ASCON sono parametrizzati tramite:

- Lunghezza della chiave  $k < 160$  bits
- Rate (dimensione del blocco)  $r$
- Numero di round interni  $a$  e  $b$

Così facendo ogni schema definisce la propria funzione di cifratura  $\epsilon_{k,r,a,b}$  e decifratura  $D_{k,r,a,b}$ . La procedura di cifratura prende in input una chiave  $K$ , un *nonce*  $N$  da 128 bit, eventuali dati associati  $A$  ed un plaintext  $P$  di lunghezza arbitraria. Restituisce un ciphertext  $C$  della medesima lunghezza del plaintext. Dunque

$$\epsilon_{k,r,a,b}(K, N, A, P) = (C, T)$$

La procedura di decifratura restituisce un plaintext se e solo se l'integrità è assicurata, ovvero previa uguaglianza del tag ricevuto e del tag ricalcolato

$$D_{k,r,a,b}(K, N, A, C, T) \in \{P, \perp\}$$

#### 2.1.2 Parametri

Per quanto riguarda i parametri suggeriti:

Nome	Algoritmo	k	nonce	tag	blocco	a	b
ASCON-128	$\epsilon, D_{128,64,12,6}$	128	128	128	64	12	6
ASCON-128a	$\epsilon, D_{128,128,12,6}$	128	128	128	128	12	8

### 2.1.3 State

Tutti i design della suite utilizzato uno state da 320 bit, gestito tramite 5 parole da 64. Lo state viene aggiornato con la funzione  $p$ , ripetuta  $a$  o  $b$  volte in base alla fase in cui ci si trova (inizializzazione, cifratura/decifratura, finalizzazione).

## 2.2 Authenticated Encryption

La modalità utilizzata da ASCON è basata sulle modalità **duplex** come *MonkeyDuplex* e *Sponge-based* design, seppur utilizzi una **inizializzazione** e **finalizzazione** (entrambe basate su chiave) più robuste. L'operazione di cifratura e decifratura avviene come in figura:

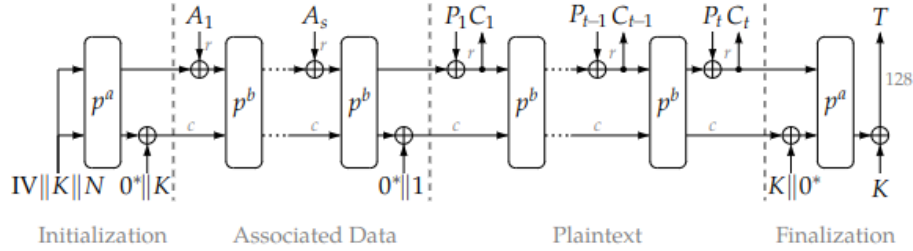


Figure 1:  $\epsilon_{k,r,a,b}$

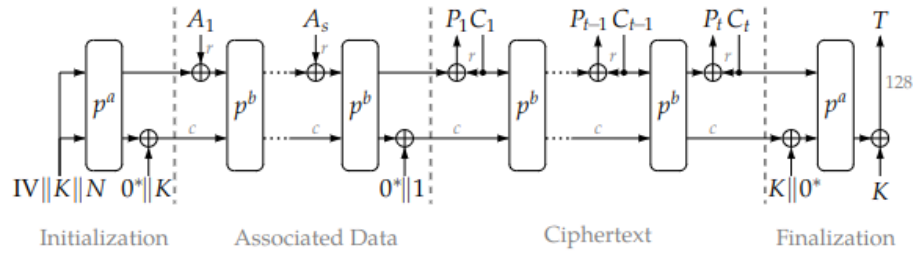


Figure 2:  $D_{k,r,a,b}$

**Inizializzazione** L'inizializzazione prevede uno state di partenza con la seguente forma:

$$S \leftarrow IV_{k,r,a,b} || K || N$$

$$IV_{k,r,a,b} \leftarrow k || r || a || b || 0^{160-k} = 80400c0600000000 \quad \text{per } ASCON - 128$$

L'inizializzazione si conclude con:

$$S \leftarrow p^a(S) \oplus (0^{320-k} || K)$$

**Dati associati** I dati associati sono processati in blocchi da  $r$  bit. L'eventuale padding viene realizzando con l'append di un 1 finale ed il minor numero di 0 necessari per raggiungere una lunghezza che sia multiplo di  $r$ . Nel caso in cui non ci dovessero essere dati associati, si passa direttamente alla fase di cifratura/decifratura. Ogni blocco di dato associato viene messo in XOR con i bit più significativi dello state e la porzione di state sostituita con il risultato. Questo implica che il tag finale dipenderà anche dal valore dei dati associati e la verifica fallirà a fronte di un tampering di questi.

Alla fine di questa fase, lo state viene messo in XOR con 1 come costante di separazione del dominio. Questa operazione è svolta come contromisura di alcuni attacchi che potrebbero cambiare il ruolo dei blocchi di plaintext e dati associati.

**Cifratura/Decifratura** Come per i dati associati il plaintext/ciphertext viene suddiviso in blocchi con eventuale 1 + padding. Durante la cifratura, ad ogni XOR lo state viene aggiornato con il risultato dell'operazione. L'ultimo blocco di ciphertext viene troncato alla lunghezza dell'ultimo blocco di plaintext senza padding, in modo tale da preservare la lunghezza originale del messaggio. Si riapplica  $p^b$  dopo ogni XOR ad eccezione dell'ultimo.

Per la decifratura invece il blocco di ciphertext viene inserito nello state ed utilizzato per effettuare uno XOR, in modo da recuperare il plaintext originale. Si riapplica  $p^b$  dopo ogni XOR ad eccezione dell'ultimo.

Si noti come in assenza di dati associati e in presenza di un singolo blocco di plaintext da cifrare, non sono necessarie altre  $p^b$  sullo state. Per prevenire lo scenario in cui lo XOR della chiave si cancelli, lo XOR viene effettuato su due parti dello state diverse.

**Finalizzazione** Durante la finalizzazione la chiave  $K$  viene XORata con lo state, per poi applicare  $p^a$ . Si estrae dunque il tag  $T$  dai 128 bit meno significativi dello state XORati con la chiave  $K$ .

## 2.3 Funzione p

La funzione  $p$  introduce le proprietà di confusione, tramite S-Box, e diffusione, tramite il linear layer. Ogni round  $p$  è un SPN-based round.

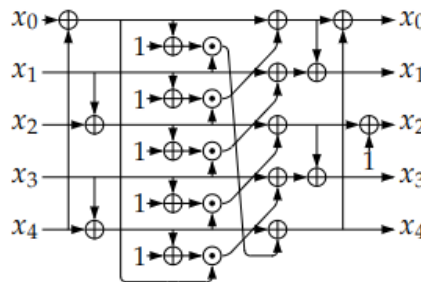
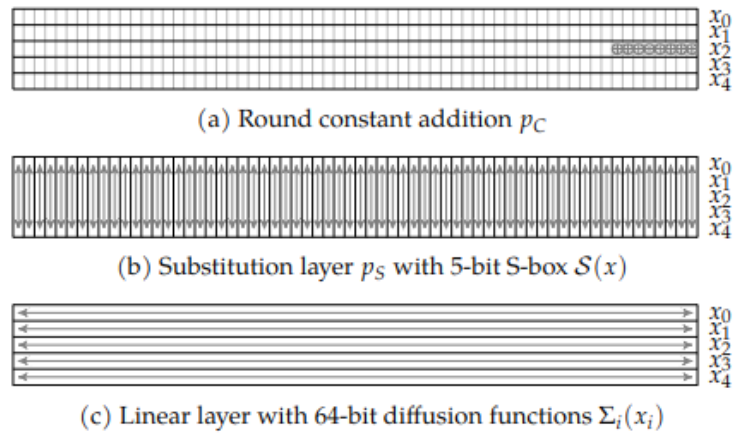


Figure 3: Approccio bitsliced per applicazione della S-Box

La prima operazione svolta è uno XOR con una costante. La costante è identificata dal numero di round in cui ci si trova. Si passa poi alla sostituzione tramite S-Box, la cui implementazione è preferibile nella sua forma *bitsliced* con operazioni effettuate sulle intere parole da 64 bit. Infine si passa al layer di diffusion, dove ad ogni parola da 64 bit si applicano degli XOR con la medesima parola ruotata di un determinato numero di bit a destra.

**Costante di round** Le costanti di round sono state scelte grandi abbastanza per evitare *slide*, *rotational*, *self-similarity* o attacchi simili. La scelta è stata effettuata in maniera semplice, ovvero incremen-

tando o meno un counter. La posizione invece è stata scelta per permettere il pipelining con le operazioni precedenti e successive (ad esempio le istruzioni successive per le S-box).

**Substitution layer** Il layer di sostituzione è l'unica porzione non lineare del round. Mixa i 5 bit che compongono una colonna nello state.

## 2.4 ASCON-128

Si tratta di un cifrario a blocchi che prevede operazioni di sostituzione e permutazione, nonché di somma e rotazioni di bit. Come accennato tutte le operazioni sono ripetute in più round ed implementate unicamente tramite bitwise operations.

### 3 Design Rationale

La suite è basata sullo **sponge design**. La funzione  $p$  è basata su SPN (Substitution-Permutation-Network). Per questo le componenti principali del cifrario sono ispirate a schemi già standardizzati.

Il substitution layer utilizza una sostituzione affine equivalente alla S-Box utilizzata in *SHA-3*. Il permutation layer utilizza funzioni lineari simili a quelle utilizzate da *SHA-2*.

Lo sponge-based design in particolare si avvicina allo *SpongeWrap* e al *MonkeyDuplex*. Lo sponge-based design ha diversi vantaggi rispetto ad altre tecniche di costruzione come i tradizionali cifrari a blocchi o le modalità hash-based:

- Ampiamente studiate ed analizzate con diverse dimostrazioni sulla loro sicurezza. Utilizzate in *SHA-3*
- Design semplice, privo di key scheduling
- I blocchi di plaintext e ciphertext possono essere computati direttamente, senza attendere per la ricezione del messaggio completo o di conoscere la lunghezza di esso.
- Utilizzo della medesima funzione sia per cifrare che per decifrare.

#### 3.1 Sponge Function

TODO

#### 3.2 MonkeyDuplex

TODO



## 4 Crittoanalisi

### 4.1 Security Claims

Tutti i design della suite offrono una sicurezza a 128 bit. Il numero di blocchi di plaintext e dati associati processati e protetti dallo schema è limitato a  $2^{64}$  blocchi per chiave, che corrispondono a  $2^{67}$  bytes (per ASCON-128). Al fine di assicurare una sicurezza a 128 bit le implementazioni devono assicurarsi che il nonce non sia ripetuto per più di due cifratura con la stessa chiave.

Inoltre il design assicura sicurezza anche di fronte ad alcuni errori implementativi, come i nonce ripetuti. Ulteriormente, anche un recupero completo di uno state durante l'elaborazione dei dati associati, del plaintext o del ciphertext non implica la possibilità di effettuare attacchi di recupero della chiave.

Non vi sono ulteriori limitazioni sulla scelta del nonce, che può essere scelto anche in maniera incrementale. Come per il resto dei cifrari, anche nel caso di ASCON osservano il ciphertext si può scoprire la lunghezza del plaintext, a meno ulteriori padding extra alla specifica.

Durante i test crittoanalitici nella competizione CAESAR tutte le analisi hanno restituito un buon margine di sicurezza, senza alcuna indicazione relativa a possibili debolezze. I miglior attacchi si sono incentrati su implementazioni con un numero di round ridotto nella fase di inizializzazione dello schema da 12 a 7 round, mantenendo comunque l'attacco lontano dall'essere una minaccia effettiva.

Lo schema è stato disegnato per assicurare una robustezza di fronte ad eventuali errori implementativi. Ad esempio, se un attacco dovesse essere in grado di recuperare uno state interno durante il processamento dei dati (ad esempio tramite un attacco side-channel), questo non permetterebbe comunque di recuperare direttamente la chiave.

Particolare attenzione all'implementazione delle S-Box: la possibilità di definire le sostituzioni realizzate tramite S-Box direttamente attraverso operazioni bitwise sullo state rendono evitabile un'operazione di table lookup. Una tale operazione risulterebbe infatti onerosa di risorse computazionali e aprirebbe le porte ad eventuali side-channel attack, dal momento in cui sarebbe possibile identificare l'istante temporale in cui il lookup avviene osservando la variazione di consumo energetico del sistema. La protezione da side-channel attack rappresenta di uno degli obiettivi principali di ASCON. A questo scopo è importante che sia semplice proteggere la S-Box. La sua implementazione permette proprio di contrastare questi attacchi.

### 4.2 Security Analysis

Seguono i report di diversi attacchi noti testati sullo schema di cifratura. Le permutazioni di ASCON non sono considerate come delle permutazioni ideali di 320 bit, ma con i parametri suggeriti permettono comunque di ottenere un generoso margine di sicurezza.

Attualmente, il miglior attacco crittoanalitico può recuperare la chiave con una complessità in tempo di  $2^{104}$  se e soltanto se l'inizializzazione è ridotta a 7 round.

#### 4.2.1 Crittoanalisi differenziale e lineare

Di seguito in figura DDT (Differential Distribution Table) e LAT (Linear Approximation Table).

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
0	32	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
6	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
7	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
8	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
9	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
a	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
b	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
c	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
d	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
e	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
f	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
10	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
11	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
12	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
13	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
14	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
15	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
17	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
18	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
19	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1a	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1b	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1c	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1d	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1e	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1f	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figure 4: DDT

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
0	16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
6	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
7	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
8	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
9	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
a	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
b	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
c	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
d	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
e	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
f	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
10	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
11	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
12	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
13	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
14	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
15	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
17	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
18	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
19	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1a	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1b	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1c	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1d	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1e	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1f	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figure 5: LAT

## 5 Protocollo di comunicazione

Si passa ora alla proposta di implementazione di un semplice protocollo di comunicazione half-duplex che fa uso dello schema appena implementato.

In particolare, si propone un protocollo di comunicazione tra due dispositivi embedded, uno dei quali funge da *master* e l'altro da *slave*. Il protocollo da cui si è stati ispirati corrisponde ad uno *Stop And Wait ARQ*: Il master invia un messaggio cifrato allo slave, che conferma la ricezione del messaggio. Il master attende la ricezione della conferma prima di trasmettere il messaggio successivo. Tecniche di controllo di errore sono implementate per rispondere ad eventuali perdite di messaggi o alterazioni del contenuto, che causerebbero un fallimento della verifica del tag.

In questa sede non si discuterà di come i due dispositivi si scambino le chiavi, ma si suppone che le chiavi siano già state scambiate in precedenza. Si suppone che entrambi i nodi, all'inizio del protocollo,

condividino una chiave segreta  $K$  ed un nonce  $N$ .

Il master trasmette un messaggio con il seguente formato:

`messageLength.nonce....message.tag`

Dove della porzione *nonce....message* ne viene trasmesso il ciphertext.

Si trasmette la lunghezza del payload, il nonce utilizzato per la cifratura, il payload effettivo ed il tag per la verifica dell'integrità del messaggio, previsto dalla specifica di ASCON. la porzione cifrata non viene trasmessa con un particolare encoding, dunque è necessario conoscere anche la lunghezza del messaggio affinché lo schema di decifratura possa operare correttamente. Il nonce utilizzato viene utilizzato in modo tale che lo slave possa verificare che il nonce utilizzato per la decifratura corrisponda a quello utilizzato per la cifratura. Si tratta di un meccanismo ulteriore oltre al tag per verificare che la decifratura sia avvenuta in maniera corretta. In questo modo il receiver può avere la certezza di essere sincronizzato con il master circa il nonce utilizzato, per poi incrementarlo per la decifratura successiva. Infine il tag, per realizzare il processo di decifratura e verificare l'integrità.

Quando il ricevente verifica, come indicato qui sopra, la corretta decifratura del messaggio, trasmette un messaggio contenente "ok" cifrato con il nonce concludendo con l'incremento del nonce.

Il master riceve l'ack, verifica la decifratura e se il messaggio corrisponde ad "ok", incrementa il nonce anche lui, per poi passare alla trasmissione successiva.

Nel caso in cui l'ack non dovesse essere ricevuto dal master, il master si occuperebbe di ritrasmettere il messaggio. Qui, due scenari:

- Il messaggio non era stato ricevuto dallo slave: entrambi sono ancora sincronizzati sul medesimo nonce, dunque ad una ritrasmissione successiva la comunicazione può riprendere
- Il messaggio era stato trasmesso, lo slave aveva inviato la conferma, che però non è stata ricevuta. Lo slave si trova ad un nonce successivo rispetto a quello del master. Lo slave, alla ricezione della ritrasmissione non sarà in grado di decifrare correttamente, dunque entra in uno stato di *retryNonce* in cui tenta la decifratura con il nonce precedente. Se la decifratura va a buon fine, lo slave ripristina il nonce al fine di sincronizzarsi con il master.

## 5.1 Implementazione

L'esempio implementativo proposto prevede l'utilizzo di due sistemi embedded ESP8266 e l'utilizzo di due moduli LoRa per trasmissione e ricezione.