SPARKS: A Serverless Protocol for Authentication and Resilient Key Sharing in UAV Networks

Timothé Pitault*
ENSEIRB-MATMECA, Bordeaux INP, France
timothepitault@gmail.com

Mauro Conti†
University of Padua, Italy
mauro.conti@unipd.it

Federico Corò
University of Padua, Italy
federico.coro@unipd.it

***Abstract*—Unmanned Aerial Vehicles (UAVs) operate under strict Size, Weight, and Power (SWaP) constraints, making traditional security mechanisms impractical. Many existing authentication protocols rely on centralized infrastructure, introducing Single Points of Failure (SPoF) and requiring continuous connectivity—unsuitable for decentralized UAV swarms. Moreover, few support the secure addition of new UAVs post-deployment. We propose SPARKS, a lightweight, serverless authentication protocol tailored for UAV swarms. Unlike prior approaches, SPARKS enables dynamic, mutual UAV-to-UAV authentication without central authorities, using only XOR operations, cryptographic hashes, and Physical Unclonable Functions (PUFs). It provides resilience against common attacks, including impersonation and device capture. Security is formally verified using the Tamarin Prover, and performance is validated on resource-constrained devices (Raspberry Pi 4 and 5). Results demonstrate that SPARKS achieves strong security guarantees and practical efficiency, making it a compelling solution for secure, flexible UAV swarm coordination.**

***Index Terms*—Unmanned Aerial Vehicles (UAVs), UAV swarms, authentication protocol, Physical Unclonable Functions (PUFs), decentralized security, lightweight cryptography, serverless authentication, dynamic enrolment, Internet of Drones (IoD), resource-constrained devices.**

I. Introduction

Unmanned Aerial Vehicles (UAVs) are increasingly used in diverse applications, from environmental monitoring to military reconnaissance. Their autonomy and deployment flexibility make them critical assets, but strict Size, Weight, and Power (SWaP) constraints limit onboard computing, memory, and energy. These limitations, along with exposure to hostile environments, necessitate lightweight yet robust security mechanisms to defend against threats like device capture, message interception, impersonation, and replay attacks.

Authentication is a key requirement, enabling UAVs to verify peer identities before exchanging sensitive data. However, traditional protocols often rely on heavy cryptography or persistent connectivity to central authorities—both unsuitable for real-world UAV deployments.

Recent efforts have explored Physical Unclonable Functions (PUFs) to address this challenge. PUFs exploit manufacturing variations to produce pseudo-unique, reproducible responses to given challenges. Various designs exist, including D-PUFs [1], Arbiter PUFs, and their XOR-based variants like FF XOR PUF [2] and K-XOR PUF [3], which increase robustness against modelling and brute-force attacks.

Another critical challenge is decentralization. Centralized schemes suffer from single points of failure and poor scalability. We aim to enable mutual authentication in UAV swarms without any central authority, ensuring resilience and flexibility, including post-deployment UAV additions.

Existing protocols [4], [5], [6], [7], [8], [9], [10] often rely on centralized infrastructure and do not support dynamic swarm composition. They are also vulnerable to systemic compromise if a single node is captured.

To address these gaps, we propose SPARKS, a lightweight and fully decentralized authentication protocol for UAV swarms. Our contributions are:

- We introduce the first decentralized UAV-to-UAV authentication scheme without centralized infrastructure, eliminating single points of failure.
- SPARKS uses XOR, hashes, and PUFs to achieve lightweight, secure authentication under SWaP constraints.
- We support secure post-deployment UAV enrolment, enabling flexible and dynamic swarm formation.
- We provide formal security verification using the Tamarin Prover and assess real-world feasibility through implementation on Raspberry Pi 4 and 5.

The rest of the paper is organized as follows: Section II reviews related work. Section III presents the threat model. Section IV details the SPARKS protocol. Section V presents our security analysis, and Section VI evaluates implementation and performance. We conclude in Section VII.

II. Related Works

Lightweight authentication has been extensively explored in IoT, especially for resource-constrained UAVs. Many protocols rely on centralized authorities. PUF-based schemes, such as those by Aman [7], Mahalat [8], and Muhal [9], show promise but generally depend on servers for key management.

PLAKE [5] combines PUFs and key exchange for UAVs, but suffers from vulnerabilities including impersonation and

*This research was supported by the Light S&T Graduate Program of the University of Bordeaux.

†Prof. Conti is also Wallenberg WASP Guest Professor at Örebro University, Sweden

979-8-3503-9281-4/25/$31.00 ©2025 IEEE

desynchronisation [11]. SecAuthUAV [4] and SLAP-IoD [6] address UAV authentication using PUFs but rely on centralized infrastructures, limiting suitability for UAV swarms. These works, however, introduce useful ideas like XOR obfuscation and CRP renewal.

Sec-PUF [10] offers efficient key updates via the Chinese Remainder Theorem, yet still depends on central coordination. However, such server-based approaches are less suitable for decentralized, peer-to-peer UAV communication scenarios.

Few works address serverless authentication. D2D-MAP [12] enables peer PUF-based authentication but remains complex and heavyweight. Ayebie et al. [13] propose mutual authentication without servers and emphasize the importance of dynamic swarm membership.

In contrast, SPARKS is fully decentralized, lightweight, and tailored for UAV-to-UAV authentication. It combines PUFs and XOR in a minimal design, eliminates central trust anchors, and uniquely supports secure, post-deployment UAV enrolment in swarms.

III. Threat & System Model

The system considered in this paper is constituted of two types of entities: UAV and a secure base station (BS). The UAVs are limited by SWaP constraints and progress in potentially hostile areas, which means they can potentially encounter adversaries on their path. Each UAV has a unique strong PUF (see Section I) resistant to tampering. As assumed in [4], [5], this paper also assumes that if a UAV is captured, an attacker cannot operate its PUF correctly or fully restore the UAV's previous functionality and continue its operations as before.

The base station, only used if a drone wants to join the swarm, is considered to be in a secure location outside the reach of attackers. It is not considered limited in calculation power or energy. The base station is equipped with a strong PUF that functions as a distinct yet tightly integrated entity, connected via a secure interface. Finally, the attacker is assumed to have the ability to eavesdrop on communications between two UAVs, intercept and potentially modify messages, and possesses substantial computational power. It is also assumed that the attacker has full knowledge of the protocol presented in this paper.

IV. Our Proposed Protocol: SPARKS

A. Overview

The proposed scheme consists of four procedures, namely, the *UAV enrolment*, the *UAV authentication*, the *additional UAV enrolment* and the *additional UAV authentication procedures*. These schemes will be detailed in the following subsections. Note that the notation used is relatively simple. This is because only two parties are considered at once to present the schemes, allowing one of them to be omitted in the notation. For example, the notation x_B signifies the number that *A* uses to create the challenge for *B*; here, *A* is omitted; however, since no other UAVs are part of the scheme, the notation remains unambiguous.

A key aspect of our protocol is the use of a three-part CRP presented as (x, C, R), inspired by [12]. Let two UAVs *A* and *B*; the value of C is obtained by passing x through a PUF, say PUF_A, and R is obtained by passing C through another, say PUF_B. Let *A* store x and R, and *B* store C. If A sends a value obfuscated by R, then only *B*, recomputing R, can recover the value. Reciprocally, only *B* was able to predict the result R of C through PUF_A, authenticating it. This is the authentication principle of our protocol.

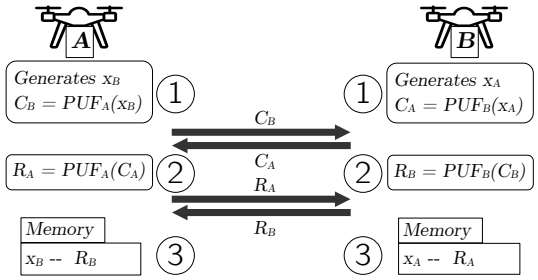


Figure 1. The UAV enrolment scheme.

B. UAV Enrolment

Before a swarm of UAVs is sent on a mission, each UAV has to undergo the enrolment procedure. This procedure allows drones to share secrets that will be used for authentication. It is therefore critical that these messages are not intercepted by an attacker. To ensure this, the enrolment should be conducted either in a secure location outside of the attacker's reach, or over a wired connection, such as while the drones are charging. The goal of this procedure is to enable each UAV of a swarm to be able to authenticate all other UAVs in the same swarm. This procedure for two UAVs, *A* and *B*, illustrated in Fig. 1, unfolds as follows: (1) Each UAV starts by generating a random number, x_A and x_B. These numbers are then processed in the PUF of each UAV to create the challenge C_A and C_B sent to the other UAV. (2) Upon receiving the challenge from their counterparts, *A* and *B* compute the responses R_A and R_B then send them back to each other. (3) Each puts and keeps in memory the random number $x_{A|B}$ they first generated and the response $R_{A|B}$ associated with their counterpart.

This procedure has to be repeated between each UAV of the swarm. Once every member of the swarm has completed the procedure, each has two values with which they can mutually authenticate other members, and one value used when other members want to mutually authenticate.

C. UAV Authentication

During the swarm mission, a UAV will need to authenticate other UAVs to determine whether they are part of their swarm or not, hence whether they can be trusted or not. The scheme represented in Fig. 2 presents the algorithm to mutually authenticate two UAVs *A* and *B*. Apart from authenticating other UAVs, some situations require them to share more information by establishing a shared key. Therefore, the optional steps required to establish a shared key (highlighted in red) are also

shown in Fig. 2. In the represented situation, *A* initiates the authentication process with *B*.

1) Simple authentication: (1) *A* initiates the authentication process by sending its *ID* and message M_0 containing a randomly generated nonce N_A used for freshness (see Section V-A2). M_0 is the XOR of N_A and C_A. (2) Upon receiving the message, *B* gets x_A from memory and construct the challenge for *A* using its own PUF. Then *B* retrieves N_A from M_0 using C_A. Finally, *B* generates a random value γ_B and constructs N_B using its PUF. This nonce is used to craft M_1 as the XOR of N_A, N_B and R_A. Finally, *B* sends its *ID*, M_1 and a hash of C_A, R_A, N_A and N_B. (3) *A* computes the response R_A to its own challenge to retrieve N_B from M_1. *A* can now verify the hash and, in case of a match, it indicates that C_A and R_A were in possession of *B*, thus authenticating it. (4) Then *A* computes the response R'_A to its future challenge N_B (see the last step). That answer is hidden in M_2 as the XOR of N_A and R'_A. *A* then sends M_2 and a hash of R_A, R'_A, N_A and N_B to *B* (5) *B* retrieves R'_A from M_2 using N_A and verifies the hash. If it found a match, it indicates that the retrieved R'_A is correct and *A* retrieved N_B from M_1, therefore, R_A was computed by *A*, authenticating it. (6) *B* then changes x_A to γ_B and R_A to R'_A in its memory. Finally, *B* sends an acknowledgement message from its *ID* and a hash of R'_A, N_A and N_B (7) If the hash matches, *A* modify C_A in its memory to be N_B.

2) Key establishment: To establish a common secret key, the previous process only needs a few extra tweaks starting from Step 4: (4) After creating the message M_2, *A* generates a random number S used in the creation of the key. *A* conceals S in M_K as the XOR of S, N_A and N_B. The key K is then derived from a HMAC Key Derivation Function (HKDF) using N_A,N_B and S. Finally, the message sent to *B* includes M_K and the hash also uses the value of K. (5) *B* retrieves S using M_K, uses it to create K with the same HKDF and verify the hash. (6) *B* adds the key K to the hash sent back to *A*. These small changes in the algorithm allow the two UAVs to share a key that can be used for the rest of the session.

D. Additional UAV Enrolment

The two procedures presented previously allow UAVs to authenticate each other and establish shared communication keys, but all drones must be present during the enrolment process. In this section we try to outgrow this limitation by proposing an additional UAV enrolment scheme.

The goal of this scheme is to allow a new drone *C* to gain the trust of *A* without having participated in the initial enrolment process.

After *A* has registered with its swarm, an additional process takes place between *A* and the base station: (1) The base station creates a list of random numbers Lx_A that is sent to its external PUF. Lx_A is kept in memory by BS. (2) The external PUF answers by sending back a list LC_A containing all the respective results of all the random numbers of Lx_A processed by the external PUF. That list is sent directly to *A*. (3) *A* creates the list LR_A that contains the respective responses to

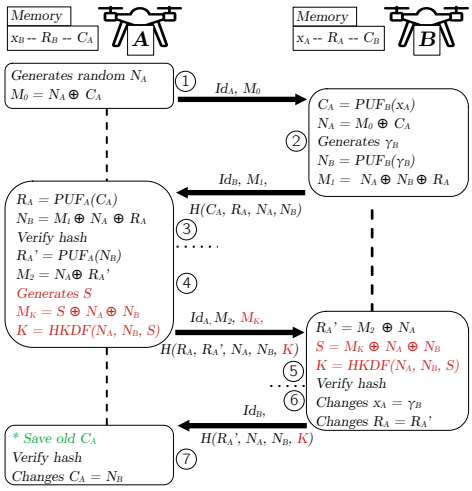


Figure 2. The UAV authentication scheme with the optional steps to establish a shared key highlighted in red.

the challenges submitted in the list LC_A. LR_A is sent back to BS. (4) The base station BS stores the list of responses LR_A in its memory.

Once this process is completed, *A* leaves the base station with its swarm. UAV *C*, having missed the swarm enrolment, must start a second process between BS and *C*, the *additional credential retrieval* (1) BS looks in its memory for Lx_A and LR_A. Once it has found those lists, it selects one random number x_A from Lx_A and the associated response R_A from LR_A. x_A is sent to the external PUF to retrieve the challenge C_A. (2) Once BS receives C_A, it is transmitted to *C* along with R_A. (3) *C* now wants to conceal R_A in such a way that a UAV capture would not reveal its value, leading to an entry gate into the trust of the swarm. To do so, *C* selects a random number X_L, generates L with its PUF, and stores the XOR result of R_A and L in a secret value S. *C* now discards R_A and L only to store C_A, X_L and S.

C has now a shared secret with *A*; *C* can predict the result of $PUF_A(C_A)$ as R_A, which is only possible if *C* is trusted by BS to obtain the secret value.

E. Additional UAV Authentication

The process of authenticating a new UAV in the swarm is analogous to the normal authenticating process. The only differences are that *C* stores the challenge differently and *A* doesn't know *C*, therefore, *A* doesn't know what challenge to expect from C. These differences only impact the first steps of the authentication. These differences are illustrated in Fig. 3, where only the first steps of communication are represented. The modifications to the authentication process can be summarized as follows: (0) This step is new to the process. As *C* initiates the connection with *A* knowing that it will not find any information about *C* in its memory. (1) *A* does not find any value associated with *C*; therefore, it initiates a new UAV authentication process by sending its id ID_A and a random nonce N_A. Note here that *A* sends N_A directly and

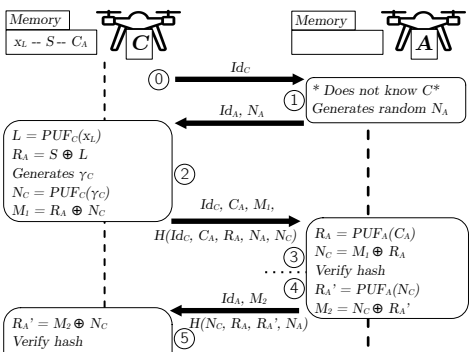


Figure 3. Changing steps of the Supplementary UAV authentication

publicly on the network because it does not yet share a secret with *C*. This step corresponds to *A* initiating the connection with *C*. (2) Upon receiving *A*'s message, *C* retrieves R_A from S by constructing L from x_L using its PUF. The rest of this step is mostly similar to the normal authentication except that M_1's content is not XORed with N_A; As the value is publicly known, it does not add any security. Also, *C* also sends C_A, as *A* does not know what challenge to expect, and the hash contains the value of *C*'s *ID*, to protect against modification of the *ID* in the message (see Section V-A3). (3) In this next step, the only difference is that *A* does not fetch the challenge from memory, nor does it use N_A to recover N_C or craft M_2. (4) *C* also does not expect *A* to craft M_2 with N_A.

The rest of the process is similar to the authentication seen in Section IV-C1. At the end of this process, *C* will discard all previous values used to authenticate *A*, since they are made for a single use, and create $x_A \coloneqq \gamma_C$ and $R_A \coloneqq R_A{}'$. *A* will keep $C_A \coloneqq N_C$ in its memory. However, the reader can remark that the final situation only allows for a one-way authentication from *A* to *C*. This is not a problem *per se*, as this situation still allows the two UAVs to authenticate with each other. A potential solution could be for *C* to authenticate again and establish a session key that could be used to share another secret in a way similar to the initial swarm enrolment process, making two-way authentication possible.

V. Security Analysis

In this section, we evaluate the SPARKS protocol from a security perspective. Our analysis is divided into two parts. First, we examine how SPARKS withstands common attack scenarios typical of UAV communication systems. Then, we present a formal security analysis to support the robustness of the protocol. All variables and notations used here refer to those previously defined in the protocol description and figures in Section IV.

A. Resilience Against Common Attacks

We assess the resilience of SPARKS against replay, impersonation, man-in-the-middle (MitM), brute-force, modelling, physical, desynchronisation, and key disclosure attacks. Each subsection highlights how design choices prevent or mitigate such threats.

1) Impersonation Attacks: An attacker cannot forge valid authentication messages without knowledge of the underlying secrets. In the basic protocol (Sec. IV-C1), attempts to impersonate *A* or *B* fail at the verification step, as the attacker lacks necessary hash inputs. In the extended protocol (Sec. IV-E), impersonating *A* may reveal C_A, but this alone does not compromise swarm security.

2) Replay Attacks: Session-specific nonces are embedded in all hash computations, rendering replayed messages invalid. As a result, each session is cryptographically bound to fresh inputs, thwarting replay attempts.

3) Man-in-the-Middle Attacks: Integrity is ensured by hashing session data, making message modification detectable. Although IDs are not always hashed, mismatched IDs result in aborted authentication, not leakage. During new UAV enrolment (Sec. IV-E), the ID is included in the hash to prevent MitM manipulation.

4) modelling Attacks: CRPs are never exposed directly; responses are masked with nonces and hashes. Even when C_A is revealed during new UAV authentication, this alone is insufficient for modelling attacks, assuming adequate bit length [5].

5) Brute-Force Attacks: With 128-bit values and at least two unknowns per hash, the effective entropy is 256 bits. Even with 2^{80} operations/sec, brute-forcing would require 2^{156} seconds for a one-in-a-million success. XORed values reduce complexity to 2^{128}, still infeasible. Using 256-bit values future-proofs the design.

6) Physical Attacks: We assume attackers cannot extract or clone PUF behaviour (Sec. III). If breached, trust is limited to UAVs linked during enrolment, restricting exposure in partially connected swarms.

7) Desynchronisation Attacks: Impersonation is already addressed; jamming attacks remain. If the message in Step 4 is jammed, both parties can safely abort. However, if the message in Step 7 is blocked, *B* updates while *A* reverts, desynchronizing credentials.

To mitigate this, if *A* does not receive an ACK after Step 7, it assumes *B* updated and proceeds accordingly. It saves the current challenge securely in memory while adopting the new one. If a subsequent authentication with *B* fails, *A* attempts re-authentication using the saved challenge. Upon success, it reverts to the prior state, restoring synchronization.

The stored old challenge is protected in memory using the same method as in additional credential retrieval (Sec. IV-D), preventing misuse through impersonation or key-recovery attacks.

8) Key Disclosure Attacks: If a session key is leaked, it only compromises that session. Future keys remain secure due to session independence.

B. Formal analysis

Formal analysis using Tamarin Prover aimed to verify the secrecy of session keys and secrets via security lemmata. Initial verification encountered non-termination due to XOR complexity [14], [15]; replacing XOR operations with

symmetric encryption yielded a tractable model. With this substitution, all lemmata were successfully verified, indicating no fundamental design flaws. However, this modified analysis does not rule out attacks exploiting combinations of captured XOR messages.

To compensate for the abstraction of XOR in the Tamarin model, a separate symbolic analysis of XOR equations was conducted. This analysis identified disjoint "leak groups," wherein leakage of one variable implies others within the same group. SPARKS was designed so that each pair of variable that could lead to impersonation were contained in different leak groups, requiring the leakage of at least two variables in different leak groups to risk an impersonation attack.

Leak group separation is designed to prevent immediate compromise from single variable leaks. However, their cyclic nature means that a leak allows tracking sessions, which can lead to a full breach upon subsequent leak (chain leak). Periodically renegotiating shared secrets using the algorithm mitigates this by decoupling sessions. While not eliminating risk, this renegotiation limits the impact of single leaks, hindering complete security breakdowns from chain leaks during vulnerable sessions.

VI. Performance

In this section, we evaluate the effectiveness and efficiency of the proposed algorithm. First, we discuss the implementation choices, followed by an analysis of the performance measurements.

A. Implementation

The algorithm was implemented in C++ and tested on Ubuntu 22.04 (x86 architecture), as well as on Raspberry Pi 4 and Raspberry Pi 5 devices, hereafter referred to as RPI4 and RPI5, respectively, running Raspbian OS 2024-11-19 with 4 GB and 8 GB of RAM, respectively. Communication between devices was achieved over Wi-Fi using standard POSIX sockets, with messages exchanged in JSON format for structured and easily parsable communication. While a PUF was not directly implemented, it was simulated using a SHA-256 function. Although SHA-256 is not considered a lightweight hash function, it offers a proven balance between security and efficiency that fits the capabilities of the target UAV platforms with moderate computational resources. Nevertheless, other secure hash functions could also be employed depending on specific constraints. Each UAV instance is initialized with a unique salt that is consistently used as input to its SHA-256 function, ensuring that the function behaves uniquely for that specific instance. It differs from a real physical PUF simply because it is software based and it can vary through time as each instantiation creates a new and different PUF. All tests and implementation scenarios are available in the gitlab repository.[1]

[1]https://github.com/mpiccoli18/SPARKS

B. Measurements

Performance evaluation focuses on three main aspects: execution time, memory usage, and communication overhead. All benchmarks were conducted between RPI4 and RPI5, with average values over 10 executions of the protocol to minimize noise. Every use of the SHA-256 function was preceded by a call to a warm-up function to configure the internal states of OpenSSL [16].

1) Execution Time: Fig. 4 shows CPU cycles for enrolment (passive and active), authentication (with and without key creation), and supplementary authentication (roles *A* and *C*). Bars distinguish between operational and idle cycles, with RPI4 and RPI5 results shown on the left and right, respectively.

Supplementary authentication is the most demanding for both platforms as one more message is required. The overhead of key creation is modest relative to the full authentication cost. A notable discrepancy appears in enrolment: active enrolment is slower on RPI4 due to slower random number generation, making RPI5's passive enrolment slower overall, as it waits for RPI4 to finish computation.

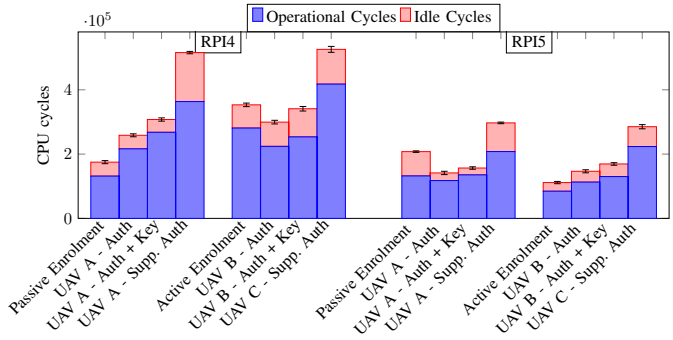


Figure 4. CPU cycles for the enrolment, authentication (Auth), authentication with creation of a key (Auth + Key) and supplementary authentication (Supp. Auth).

Fig. 5 shows the time costs. Since full enrolment requires both active and passive roles, the total time per side lies between the max and sum of both, depending on parallelism. This results in approximately 0.330 ms for RPI4 and 0.133 ms for RPI5, comparable to supplementary authentication.

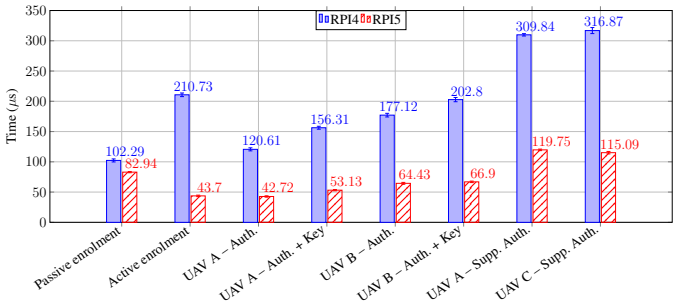


Figure 5. Histogram showing the processing time (in microseconds) for the different authentication schemes on RPI4 and RPI5.

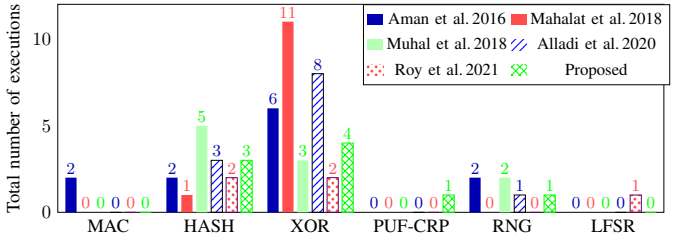
2) Memory Usage: Table I records the maximum amount of RAM used measured during the execution of the enrolment

and authentication scheme. The maximum reported memory footprint of approximately 219 KB includes the use of OpenSSL's implementation of SHA-256, which, while convenient for testing on Raspberry Pi devices, are not optimized for low-memory environments. We anticipate a significant reduction in memory usage when replacing these components with lightweight cryptographic libraries or minimal, dedicated implementations tailored for embedded systems. This optimization is especially relevant for deployment on resource-constrained devices, such as microcontrollers.

Device	Enrolment		Authentication	
	Passive	Active	UAV *A*	UAV *B*
RPI4	217,424	217,424	217,464	217,984
RPI5	217,424	217,424	219,784	217,984

Table I
MEMORY OVERHEADS (IN **KB**) DURING ENROLMENT AND AUTHENTICATION.

(a) Performance comparison of IoT device



(b) Performance comparison of Server (or other IoT node)

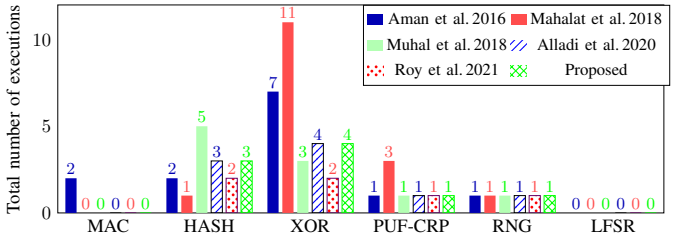


Figure 6. Number of MAC, HASH, XOR, RNG and LSFR operation executed and count of CRP needed for an authentication.

3) Number of Operations: To further evaluate SPARKS against related authentication and key exchange protocols, we reproduced and updated a comparative figure originally introduced by [5]. The plot in Fig. 6 illustrates the number of cryptographic and arithmetic operations executed by each protocol on both the IoT node and the server side. As the figures show, SPARKS maintains a balanced computational profile across all operations. Unlike other protocols, SPARKS requires the use of a PUF-CRP on both the IoT device and the server. However, different portions of the CRP are stored separately on each device.

VII. CONCLUSION

We introduced SPARKS, a lightweight decentralized protocol that allows mutual authentication of UAVs in swarms without central infrastructure. We also introduce an optional extension for dynamic enrolment, which relies on a server.

SPARKS efficiently counters threats like impersonation and capture, and securely integrates new UAVs post-deployment. Validated empirically and formally, it meets swarm security needs on resource-limited devices.

Future work may focus on further optimizing memory usage, adapting to even stricter SWaP constraints, and exploring alternative cryptographic primitives better suited for UAV limited hardware.

REFERENCES

[1] S. Sutar, A. Raha, D. Kulkarni, R. Shorey, J. Tew, and V. Raghunathan, "D-puf: An intrinsically reconfigurable dram puf for device authentication and random number generation," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 1, pp. 1–31, 2017.

[2] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 13, no. 10, pp. 1200–1205, 2005.

[3] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *annual design automation conference*, 2007, pp. 9–14.

[4] T. Alladi, G. Bansal, V. Chamola, M. Guizani *et al.*, "Secauthuav: A novel authentication scheme for uav-ground station and uav-uav communication," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 15 068–15 077, 2020.

[5] S. Roy, D. Das, A. Mondal, M. H. Mahalat, B. Sen, and B. Sikdar, "Plake: Puf-based secure lightweight authentication and key exchange protocol for iot," *IEEE Internet of Things Journal*, vol. 10, no. 10, pp. 8547–8559, 2022.

[6] S. Yu, A. K. Das, Y. Park, and P. Lorenz, "Slap-iod: Secure and lightweight authentication protocol using physical unclonable functions for internet of drones in smart city environments," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 10, pp. 10 374–10 388, 2022.

[7] M. N. Aman, K. C. Chua, and B. Sikdar, "Position paper: Physical unclonable functions for iot security," in *ACM international workshop on IoT privacy, trust, and security*, 2016, pp. 10–13.

[8] M. H. Mahalat, S. Saha, A. Mondal, and B. Sen, "A puf based light weight protocol for secure wifi authentication of iot devices," in *International symposium on embedded computing and system design (ISED)*. IEEE, 2018, pp. 183–187.

[9] M. A. Muhal, X. Luo, Z. Mahmood, and A. Ullah, "Physical unclonable function based authentication scheme for smart devices in internet of things," in *IEEE International Conference on Smart Internet of Things (SmartIoT)*. IEEE, 2018, pp. 160–165.

[10] W. Lalouani, "Sec-puf: Securing uav swarms communication with lightweight physical unclonable functions," in *2023 19th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2023, pp. 286–291.

[11] S.-W. Lee, M. Safkhani, Q. Le, O. H. Ahmed, M. Hosseinzadeh, A. M. Rahmani, and N. Bagheri, "Designing secure puf-based authentication protocols for constrained environments," *Scientific Reports*, vol. 13, no. 1, p. 21702, 2023.

[12] K. Lounis, S. H. Ding, and M. Zulkernine, "D2d-map: A drone to drone authentication protocol using physical unclonable functions," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 4, pp. 5079–5093, 2022.

[13] E. B. Ayebie, K. Bou-chaaya, and H. Rais, "A new efficient puf-based mutual authentication scheme for drones," in *International Conference on Risks and Security of Internet and Systems*. Springer, 2023, pp. 67–84.

[14] S. Kremer, S. Meier, B. Schmidt *et al.*, *The Tamarin Prover Manual: Book Version 0.9*, 2023, https://tamarin-prover.com/book/index.html. [Online]. Available: https://tamarin-prover.com/book/index.html

[15] Tamarin Prover Developers, "Tamarin prover manual," https://tamarin-prover.com/manual/index.html, 2023, accessed: 2025-04-21.

[16] OpenSSL Project, "Api design considerations: Initialization functions hurt," n.d., accessed: 2025-04-17.