A Time-Bound Continuous Authentication Protocol for Mesh Networking

Selina Shrestha*†, Martin Andreoni Lopez†, Michael Baddeley†, Sami Muhaidat*, and Jean-Pierre Giacalone†
*Khalifa University, Abu Dhabi, United Arab Emirates {sami.muhaidat}@ku.ac.ae
†Secure Systems Research Center (SSRC), Technology Innovation Institute (TII)
Abu Dhabi, United Arab Emirates
{selina, martin, michael, jean-pierre}@ssrc.tii.ae

***Abstract*—This paper proposes a novel lightweight authentication protocol for fast and efficient continuous authentication of constrained Internet of Things (IoT) mesh network devices. An initial static authentication takes place at the beginning of a session, during which the secret is securely shared between two parties. Once the session is established, the continuous authentication scheme generates time-bound tokens using the shared secret, a time-varying component, and a random value. These are used to verify the identity of the connected device in fixed time intervals. These time-bound tokens, which only remain valid for a predetermined time frame and ensure the continuity aspect of the mechanism, can be linked back to the original secret at the server for verification. Therefore, lightweight continuous authentication is achieved using only a few low complexity cryptographic operations, like hash and MAC, without the need to perform costly cryptographic operations. In case of failure, the node is temporarily blocked for an exponential function period of past occurrences of failure. Performance evaluation over a mesh routing protocol shows that our proposal fulfills lightweight and low bandwidth constraining requirements while satisfying the security requirements of an authentication scheme.**

***Index Terms*—IoT, Mesh Networks, Distributed Authentication, Continuous Authentication**

I. Introduction

The advent of the Internet of Things (IoT) has enabled connection and communication between devices, without the need for human intervention. Mesh networking, in particular, has been a key enabler of IoT, particularly within commercial and industrial sensor/actuator networks and, more recently, within dynamic, high-mobility networks such as those created by Unmanned Aerial Vehicles (UAVs) [1]. While this has enabled automated connectivity and data transfer mesh networking concepts bring added complexities to various security threats such as data tampering, hijacking attack, denial of service, etc. Thus, maintaining security and privacy in an autonomous IoT mesh network is a key challenge. Additionally, since many IoT mesh devices are resource-constrained, security mechanisms must be lightweight.

Authentication is a critical aspect of security that protects the network from unauthorized access to resources and attacks by illegitimate parties [2]. Most of the early works on authentication [3, 4, 5] focus on the initial static authentication during the establishment of the network, in which the involved parties submit their credentials such as certificates and signatures for verification by an authority. Once the parties are authenticated, communication can begin and the participating nodes' identities are not rechecked throughout the session. Unfortunately, this method leaves the network vulnerable to security attacks while the session is active. Therefore, a mechanism to *continuously* authenticate the identity of the participating nodes throughout the session is needed.

Continuous authentication ensures that a node's identity is still the one authenticated at the beginning of the session by periodically repeating the authentication process during an active communication session. Additionally for resource constrained IoT mesh devices the authentication mechanism must involve lightweight, low complexity computations that do not result in high energy consumption and must not add significantly to the bandwidth consumption of the underlying communications channel. Early works for initial/mutual authentication involve the transmission of certificates and computationally expensive cryptography operations. However, this approach is not amenable to continuous authentication [6], highlighting the need for a separate lightweight and fast authentication mechanism to verify the node's identity repeatedly during the session after the static authentication performed at the beginning of the session.

Most existing works are directed towards U2D authentication, which exploit user-specific human biometric features such as gesture pattern [7, 8], keystroke analysis [9, 10, 11] and touch-screen pattern analysis [12, 13] to repeatedly verify identity of the user using the device. These methods, however, are user and application-specific and cannot be used to authenticate the connected device irrespective of the application used. Device-to-device continuous authentication to authenticate the identity of the connected device has been proposed in a few works [6, 14] using lightweight hash and Message Authentication Code (MAC) operations for continuous authentication. While a single authentication requires eight hash operations and two MAC operations [14], the number addition and multiplication computations for a single authentication are proportional to the total number of authentications required [6]. Thus, it is possible to reduce the number of computations and, therefore, lower the authentication mechanism's complexity.

This work proposes a lightweight continuous authentication protocol for resource constrained IoT mesh devices and is an extension to the work presented in our positional paper [6]. We assume that the server is already established within the mesh, will verify a client trying to join, and will then authenticate continuously. Whenever a new node tries to enter the network, a static mutual authentication occurs between that node and a server node. Once the session is active, the client is assumed to be continuously authenticated periodically. The authentication period and a secret key (a random number that is unique to a particular session) are agreed upon and securely shared between the server and client during the static mutual authentication. For a new session, a fresh arbitrary value is set as the secret. The client generates time-bound authentication tokens (known as shares) for each continuous authentication; using the shared secret, another random number, and the time flag. At the server, these tokens can be linked back to the original secret. In this manner, the client is authenticated without the need to perform costly public/private key cryptography operations. These shares/tokens are bound as function of time so that a given share is valid only for a predetermined time slot, ensuring the 'continuity' aspect of the authentication quickly and efficiently. The computations involved comprise one hash operation and one MAC operation at client and server, along with two additions at client and two subtractions at the server, which is lower than computations in existing works. Additionally, the protocol proposes an exponential backoff mechanism to block a node in case of authentication failure for a period that depends exponentially on the number of times the client has failed. Hence, a fraudulent node gets blocked indefinitely, while a node that failed due to temporary issues gets unblocked at a specific time.

The major contributions of this paper over existing works are: *(i)* A novel lightweight authentication mechanism with fewer computations; *(ii)* a complete protocol for continuous authentication using the proposed mechanism; and *(iii)* An exponential backoff mechanism to temporarily block a node in case of failure.

II. Related Work

This section presents an overview of works and literature related to continuous authentication in IoT networks. The works have been grouped into *user-to-device (U2D)* and *device-to-device (D2D)* categories.

U2D continuous authentication schemes utilize user-specific features such as gesture pattern, keystroke analysis, and touchscreen pattern analysis along with some data mining techniques or machine learning-based classifiers to identify the user. Shimshon et al. [9] collect multiple keystrokes of a genuine user to create corresponding feature vectors as the reference base during the initial authentication. Then, continuous authentication converts newly generated keystrokes into feature vectors and compares them with the reference base to verify the user's identity. Shen et al. [15] use dynamic patterns of mouse usage by a genuine user to continuously verify the user identity. Also, Bailey et al. [16] performed multi-modal behavioral biometrics, including combined patterns of the keyboard, mouse, and Graphical User Interface (GUI) interactions of the genuine user to authenticate it with greater accuracy. Smartphone users' physical activity patterns given by accelerometer, gyroscope, and magnetometer sensors along with an Support Vector Machine (SVM) classifier to authenticate the user is used in [7]. Similarly, the UAV operator's joystick patterns along with a random forest classifier to authenticate the operator's identity were used by [8]. Acar et al. [10] on the other hand, rely on the sensor-based keystroke dynamics acquired through the built-in sensors of a wearable device while the user is typing and uses a neural network for user authentication. Finally, six types of touch gestures along with voice commands to protect the privacy of the owner of wearable glasses [12], while swiping gestures, typing patterns, and the phone movement patterns observed during typing or swiping [13] are used for user authentication.

While U2D approaches ensure that the user of a device is the same user that was initially authenticated, D2D authentication mechanisms ensure that the device communicating over the network is the same device authenticated at the beginning. Chuang et al. [14] proposed a continuous authentication protocol between a sensing and gateway device by using tokens that depend on dynamic features like remaining battery capacity. Bamasag and Youcef-Toumi [6] proposed an authentication mechanism based on a secret sharing mechanism, where a secret is agreed upon during the initial authentication and shares of the secret generated using Shamir's secret sharing algorithm [17] are used as authentication tokens to continuously authenticate the connected devices. Both works used light cryptographic operations such as hash and MAC. Table I shows the computations required for both protocols to perform a single authentication where k is the total number of authentications during the session.

Our protocol builds upon the work from Bamasag and Youcef-Toumi [6]. We improve this approach as follows:

Authentication mechanism. Bamasag and Youcef-Toumi [6] compute the share as $secret + timeflag + f(x)$ where $f(x) = a1 * x + a2 * x^2 + \ldots + ak * x^k$, which requires k multiplications and additions per authentication as well as k random number generation at the beginning, where k is the number of authentications that need to be performed during a session. In contrast, our work computes the share as $secret + timeflag + x$, where x is a random number. This requires *only* two additions and one random number generation per authentication, where k is the total random numbers generated.

Authentication period. The Bamasag and Youcef-Toumi [6] work allowed for finite k number of authentications during a session. Thus, k had to be known at the beginning of the protocol. In our work, the number of authentications is unneeded to be predefined at the beginning of the session. We only define an authentication period.

Error checking. We added a Cyclic Redundancy Check (CRC) for error-detecting in protocol exchanged messages.

Exponential backoff. Bamasag and Youcef-Toumi [6] terminated the session when the authentication failed. Our work allows us to block the client temporarily.

Server-initiated authentication. In our work, authentication is initiated periodically by the server (verifier) and not the

TABLE I: Computational cost comparison between proposed protocol and related works.

Work	Phase	Hash	MAC	Random Num	Mult	Exp	Add/Subt
Our work	Client	1	1	1	-	-	2
Our work	Server	1	1	-	-	1 (if fail)	2
[6]	Claimer	1	1	k (once)	k-1	-	k+2
[6]	Verifier	1	1	-	-	-	2
[14]	Sensor	5	1	1	-	-	-
[14]	Gateway	3 or 4	1	-	-	-	-

client (claimer). In [6], it is initiated by the claimer (client). By initiating authentication from the server-side, we can detect if a client does not participate in authenticating itself on time. **Synchronization.** Finally, the Bamasag and Youcef-Toumi [6] work relied on time synchronization between server and client to check message freshness. We have replaced the need for synchronization by initiating the authentication at the server and only using the server's local time as a reference to check message freshness.

III. Proposed Protocol

As shown in Figure 1, in the beginning, mutual authentication is done during which a $secret$ unique to the session as well as the continuous authentication period T_{cont} is shared. Once a session is established, the client is authenticated every T_{cont} seconds. For every i^{th} authentication, the server sends a request for authentication, after which the client generates an authentication message comprising a share u_i, a share authenticator sa_i, and the message payload m_i. The authentication message is sent to the server for verification, and the server performs multiple steps to verify the message. The authentication result is then communicated to the client, and the client is blocked for $backoff\_period$ if the authentication fails. Details of steps carried in the protocol for every i^{th} authentication are given below.

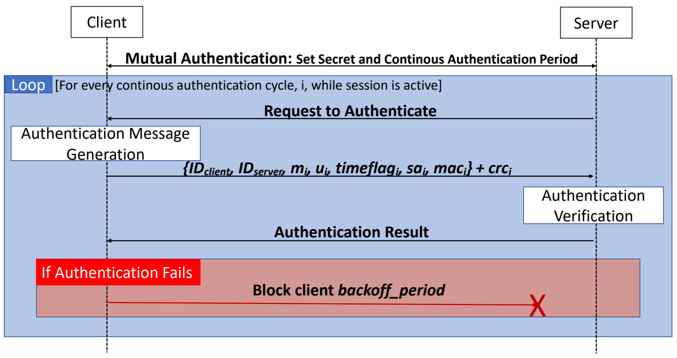
Client
Server
Mutual Authentication: Set Secret and Continous Authentication Period
Loop
[For every continous authentication cycle, i, while session is active]
Request to Authenticate
Authentication Message Generation
{ID_client, ID_server, m_i, u_i, timeflag_i, sa_i, mac_i} + crc_i
Authentication Verification
Authentication Result
If Authentication Fails
Block client backoff_period
X

Fig. 1: Block diagram of continuous authentication protocol

Step 1: Authentication Initiation (Server Side). The server initiates the authentication by sending a request to the client to authenticate itself. The timestamp at which the request is sent is stored as $start\_time$. The client is expected to return the authentication message within a certain time, $time\_margin$, which is less than T_{cont}. If the client fails to do so, its authentication fails, and **Step 4 - (S4)** is executed.

Step 2: Authentication Message Generation (Client-Side). Once the server sends an authentication request, and it is received by the client, an authentication message is generated according to Algorithm 1. To compute the time-bound token/ share, u_i, first a random number x_i is selected. Then, u_i is computed as shown in step 2 of Algorithm 1, where $timeflag_i$ = i. All used shares are stored at the client to ensure that the newly generated share has not been used in previous authentication cycles during the session. A different value for x_i is selected until a fresh u_i is generated. u_i is then appended to the list of used shares for future authentication cycles. For the computed share u_i, a share authenticator, sa_i, is computed to facilitate its authentication at the server. Then, the client generates the authentication message comprising the client ID ID_c, server ID ID_s, message payload m_i, share u_i, share authenticator sa_i and i^{th} time flag $timeflag_i$. Additionally, to ensure the integrity and source of origin, a Message Authentication Code (MAC) of (ID_c,ID_s, m_i, u_i, $timeflag_i$) using the $secret$ key is generated. The use of $secret$ for macing further ensures that the share and time flag received were generated by the valid client. The MAC, mac_i is added to the message fields to generate the authentication message. Finally, to detect any transmission loss at the server-side, Cyclic Redundancy Check bits, crc_i, are calculated for $message_i$ and appended to it. $message_i$ + crc_i is then sent to the server for verification.

Algorithm 1 Authentication Message Generation (Client Side)

```
1: Select random number $x_i$
2: Compute share, $u_i$ = $secret + timeflag_i + x_i$
3: Compute share authenticator, $sa_i$ = Hash($x_i$)
4: Compute $mac_i$ = HMAC$_{secret}(ID_c, ID_s, m_i, u_i, timeflag_i)$
5: $message_i$ = $\{ID_c, ID_s, m_i, u_i, timeflag_i, sa_i, mac_i\}$
6: Compute $crc_i$, cyclic redundancy check bits for $message_i$
7: Send $message_i$ appended with $crc_i$
```

Step 3: Authentication Verification (Server Side). Once the authentication message is received at the server, it is verified in 5 steps. *Verification V-1: Perform CRC to check data integrity*. As described in Algorithm 2, if the CRC check is successful, the server will move to the next verification step *V-2*. Otherwise, it will ask the client to resend the message for up to 5 times. If the CRC fails for more than 5 times, *V-1* will report authentication failure and the server will move to **S4**. This has been employed to avoid denial of service in case a fraudulent node keeps sending fake messages.
Verification V-2: Check message freshness. To check the freshness of the received message, the server compares the current

Algorithm 2 V-1: Perform CRC to check data integrity

```
1: Perform CRC check on received message
2: if pass then
3:     $num\_crc\_fails$ = 0
4:     Continue to next verification V-2
5: else
6:     $num\_crc\_fails$ = $num\_crc\_fails$ +1
7: end if
8: if $num\_crc\_fails <= 5$ then
9:     Ask client to resend message
10: else
11:     Return authentication failed
12:     Go to Step 4
13: end if
```

timestamp (at which the message is received) with $start\_time$ (at which authentication was requested), using Algorithm 3. If the difference is less than or equal to $time\_margin$, the message is considered fresh and the server moves to *V-3*. Otherwise, the message is considered stale, an authentication failure is reported and the server moves to **S4**.

Algorithm 3 V-2: Check message freshness

```
1: if current timestamp - $start\_time <= time\_margin$ then
2:     Message is fresh
3:     Continue to next verification V-3
4: else
5:     Message is stale
6:     Return authentication failed
7:     Go to Step 4
8: end if
```

Verification V-3: Check if received share u_i is fresh. To be resilient to packet spoofing and impersonation attacks using previously transmitted shares, the server must discard duplicate shares and strictly accept fresh shares that have not been used previously during the session. Thus, the shares received during the session are stored in the server, and any newly received share is checked if it is present in the list of used shares. As described in Algorithm 4, if the share was not received previously, the share is deemed to be fresh and is appended to the list of received shares, and the server moves to V-4. Else, the share is not fresh, and the authentication failure is reported whereby the server moves to **S4**.

Algorithm 4 V-3: Check if received share u_i is fresh

```
1: if $u_i$ has not been received previously during the session then
2:     Share is fresh
3:     Continue to next verification V-4
4: else
5:     Share is not fresh
6:     Return authentication failed
7:     Go to Step 4
8: end if
```

Verification V-4: Compute fresh MAC to verify received u_i and $timeflag_i$. A fresh MAC mac_i' of message m_i, share u_i and $timeflag_i$ received from the client is computed at the server using the $secret$ known to the server and is compared to received mac_i as shown in Algorithm 5. If the values match, it implies that u_i and $timeflag_i$ were sent by a genuine client (i.e. MAC was done using the correct $secret$ key at the client) and that they unchanged during the transit. So, the server moves to the next verification *V-5*. If the MACs unmatched, an authentication failure is reported and the server moves to **S4**.

Algorithm 5 V-4: Compute fresh MAC to verify received u_i and $timeflag_i$

```
1: Compute fresh mac as:
2:     $mac_i' = HMAC_{secret}(ID_c, ID_s, m_i, u_i, timeflag_i)$
3: if $mac_i' = mac_i$ then
4:     $u_i$ and $timeflag_i$ were generated by valid client and they
5:     were not altered during transmission
6: else
7:     Return authentication failed
8:     Go to Step 4
9: end if
```

Verification V-5: Compute fresh share authenticator to authenticate received share. A fresh share authenticator for the received share u_i is computed at the server using received u_i, $timeflag_i$ and known $secret$ and it is used for share authentication according to Algorithm 6. It is checked if the freshly computed share authenticator sa_i' is equal to the received share authenticator sa_i. If the check returns true, it is verified that the received share is associated with the valid secret. The client is therefore considered authentic and an authentication pass is reported for **S4**. Else, if the check is false, the client is not authentic and an authentication failure is reported for **S4**.

Algorithm 6 V-5: Compute fresh share authenticator to authenticate received share

```
1: Compute fresh share authenticator as:
2:     $sa_i' = Hash(u_i - secret - timeflag_i)$
3: if $sa_i' = sa_i$ then
4:     Share is authentic i.e. client is authentic
5:     Return authentication passed
6: else
7:     Return authentication failed
8:     Go to Step 4
9: end if
```

Step 4: Generate Authentication Result. According to the output of verification in **Step 3**, a result comprising $\{authentication\_result, backoff\_period\}$ is compiled and sent to the client. The $authentication\_result$, generated according to Algorithm 7 denotes the result of the authentication ("pass" or "fail") while $backoff\_period$ denotes the period for which the client is blocked (if authentication fails).

If authentication passed: The number of consecutive failures, $num\_of\_failures$ and $backoff\_period$ are reset to 0 and $authentication\_result$ is set to"pass". The message ˜pass", 0} is sent back to the client as the authentication result.

If authentication failed: The $num\_of\_failures$ is incremented by 1 and $backoff\_period$ is computed according to step 8 of Algorithm 7. So, the backoff period increases exponentially as the consecutive number of authentication failures increases. The base of the exponential is selected to be max(T_{cont}, 2) to avoid diminishing exponential value when T_{cont} is less than 1. The message {"fail", $backoff\_period$} is sent back to the client as the authentication result and messages from the client are not accepted at the server for the duration of $backoff\_period$.

Algorithm 7 Generate Authentication Result

```
1: if authentication passed then
2:     Reset $num\_of\_failures$ = 0
3:     Reset $backoff\_period$ = 0
4:     $authentication\_result$ = “pass”
5:     Send {“pass”, 0} to client
6: else if authentication failed then
7:     $num\_of\_failures$ = $num\_of\_failures$ + 1
8:     $backoff\_period = (max(T_{cont}, 2))^{num\_of\_failures}$
9:     $authentication\_result$ = “fail”
10:    Send {“fail”, $backoff\_period$} to client
11:    Do not receive message from client for $backoff\_period$
12: end if
```

IV. Results

A. Security Evaluation

We evaluate our proposed protocol against its resilience to various security attacks, and describe how different security requirements are satisfied. Raspberry-pi 4B boards were used for the evaluation, one is a server and the other clients. Each board uses a Broadcom BCM2711 (ARM v8) 64-bit SoC 1.5GHz and 8GB of RAM. The boards run Ubuntu 20.04 and execute B.A.T.M.A.N IV mesh routing protocol on the top.

1) Message source authentication: MACing share and timeflag using secret key ensure that the received share and timeflag were sent by an authentic client. Then, using the received share, timeflag, and the secret known only to server and client, a fresh share authenticator is computed at the server. Matching this with the received share authenticator assures that the authentication message originated from a valid client and the network is secure from a Man-in-the-Middle attack.

2) Continuous authentication: The client identity is continuously authenticated during the session by periodically requesting authentication messages from it. Each authentication message carries a unique share that is specific to the given time frame and is associated with the shared secret. The ‘continuity’ aspect is achieved by making the shares a function of timeflag such that each share is only valid for a predetermined timeslot. Thus, the server performs two verification to ensure continuous authentication, computing: **i)** MAC using the secret key to verify that received timeflag was generated by a valid client, and **ii)** new share authenticator to verify that the newly received share with received timeflag belongs to the shared secret.

3) Data integrity: The integrity of data is verified in two steps: **i)** Performing CRC ensures that the data has not been lost during transmission, and **ii)** Computing MAC checks that the message, share, and timeflag have not been altered during the transmission.

4) Confidentiality of the Secret: The time-varying shares are generated by adding the timeflag and a random number x to the secret. During transmission, the share, timeflag and hash of x (shares authenticator) are sent. Since x cannot be constructed from its hash, and since its value cannot be speculated because it is random, the secret cannot be determined using the share and the timeflag. Thus, an eavesdropper cannot reconstruct the secret from the transmitted values. Also, because x is random, inference about the secret cannot be drawn by studying the previously transmitted shares. Additionally, a new secret is established between the server and the client in future sessions, so that shares used in previous sessions do not compromise the security of secrets used in future sessions. This prevents the network from eavesdropping attacks.

TABLE II: CPU consumption at server for varying number of client connections

No. of clients	Avg % CPU consumption	Max % CPU consumption
1	0.74%	20%
5	2.49%	33.3%
10	4.27%	33.3%

5) Access Control: The protocol ensures that messages are received only from a valid client. If the authentication fails, the client is blocked for a certain period, which increases exponentially with the number of authentication fails. Thus, it prevents a denial of service attack.

6) Freshness: Freshness is warranted by checking that the shares received are fresh and that the messages are received promptly. Any failure results in blocking the client for a period that is exponential to the number of authentication fails This protects the protocol against the replay attack.

B. Performance Evaluation

1) Computation Cost: To evaluate the computation cost of the proposed protocol, the number of different computation operations required for each authentication cycle was derived and compared with related works as shown in Table I, where it can be seen that our work requires only *one hash* and *one MAC* operation at the client and server each (as per [6]), while [14] requires a total of 8 or 9 hash operations. Additionally, while the number of multiplications and additions/subtractions in [6] depends linearly on k, the total number of authentications required in the session, the proposed protocol only requires two additions/subtractions each at client and server respectively. Therefore, the proposed protocol can be considered computationally more efficient or more lightweight than its counterparts. Additionally, the CPU consumption at both client and server during the authentication was also observed to evaluate the computation cost of the protocol. On the *Client side*: Five measurements were taken for ten authentication cycles of period two seconds. The average % CPU consumption was found to be 0.68% with a maximum % CPU consumption of 13.3%. In addition, on the *Server side*: The CPU consumption on the server-side was measured for a varying number of client connections. Five measurements each were taken for 1, 5, and 10 simultaneous client connections. The results are documented in Table II. The average % CPU consumption is fairly low and increases slightly as the number of clients increases. The maximum % CPU consumption does not exceed beyond 33.3% for up to ten clients.

2) Communication Cost: The communication cost is measured by the average size of the authentication message transmitted during each cycle of continuous authentication. 10 measurements for average authentication message size during a session with varying numbers of authentication cycles were taken. The average authentication message size was obtained to be 404.6 bytes.

3) Storage Cost: Storage cost is given by the cost of storing all the previously used shares in the client and previously received shares in the server. The share storage cost was measured for sessions with a varying number of authentications. The result is shown in Figure 2.

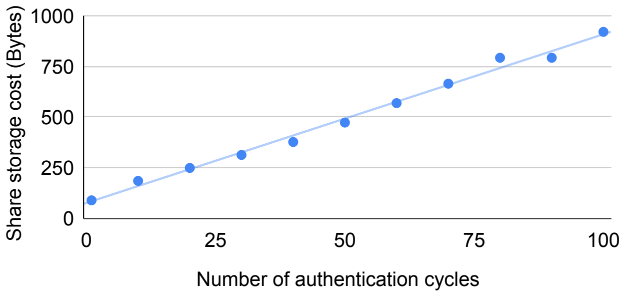

Fig. 2: Share storage cost for varying number of authentication cycles

The storage cost increases almost linearly with the number of authentications performed. The linear equation in (1) best describes the trend for share storage cost shown in Figure 2. Thus, the additional storage cost per authentication cycle is approximately 8.33 bytes.

$$storageCost = 75.15 + 8.33 * num\_of\_authentications \quad (1)$$

Additionally, the percentage memory consumption by the authentication protocol was also measured at both client and server. The memory consumption in the server was observed for varying the number of client connections: 1,5, and 10. The percentage memory consumption by the program was found to be a constant of 0.3% for all the cases. Similarly, the memory consumption in the client was tested by varying the number of authentications from 1 to 100 in steps of 10. The percentage memory consumption by the client program was also found to be a constant 0.3% for all the cases.

V. Conclusion

A lightweight continuous authentication scheme has been proposed using a shared secret, time-dependent variable, and a random number to generate time-bound authentication tokens called shares for continuous authentication during a session. An overall protocol has been proposed using the presented authentication scheme, whereby any node that fails to authenticate itself is blocked for a period depending on its record of failures. The proposed protocol has been evaluated based on its security performance and has been found to meet the security requirements of a continuous authentication scheme. In addition, the proposed protocol is also found to involve fewer hashes, MAC, and arithmetic computations comparing with existing works, which indicates that it is comparatively more lightweight and therefore more suitable for fast and efficient authentication of nodes at any point in time. Moreover, the protocol generates a reasonably light authentication message of around 405 bytes, which makes it bandwidth-efficient, while only adding a small storage cost of around 8 bytes per share. Therefore, the proposed protocol meets the lightweight and bandwidth-efficient requirements of continuous authentication of nodes in a mesh network. Future enhancements to the protocol may include encryption of messages transmitted under the protocol for an added layer of security. Another enhancement may be to explore the use of dynamic network parameters in the generation of authentication tokens.

References

[1] M. Andreoni Lopez, M. Baddeley, W. T. Lunardi, A. Pandey, and J.-P. Giacalone, "Towards secure wireless mesh networks for uav swarm connectivity: Current threats, research, and opportunities," in *Proceedings of the* 17^{th} *International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Jul. 2021.

[2] M. Ouaissa, M. Houmer, and M. Ouaissa, "An enhanced authentication protocol based group for vehicular communications over 5g networks," in *2020 3rd International Conference on Advanced Communication Technologies and Networking (CommNet)*, 2020, pp. 1–8.

[3] J. Liu, Y. Xiao, and C. P. Chen, "Authentication and access control in the internet of things," in *Distributed Computing Systems Workshops*, 2012.

[4] W. Liu, Qin, "A lightweight rfid authentication protocol based on elliptic curve cryptography," *Journal of Computers*, vol. 8, no. 11, 2013.

[5] A. A. Yavuz, "An efficient real-time broadcast authentication scheme for command and control messages," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 10, 2014.

[6] O. O. Bamasag and K. Youcef-Toumi, "Towards continuous authentication in internet of things based on secret sharing scheme," in *Workshop on Embedded Systems Security*. Association for Computing Machinery, 2015.

[7] M. E. ul Haq, M. Awais Azam, U. Naeem, Y. Amin, and J. Loo, "Continuous authentication of smartphone users based on activity pattern recognition using passive mobile sensing," *Journal of Network and Computer Applications*, vol. 109, 2018.

[8] A. Shoufan, "Continuous authentication of UAV flight command data using behaviometrics," in *IFIP/IEEE International Conference on Very Large Scale Integration*, 2017.

[9] T. Shimshon, R. Moskovitch, L. Rokach, and Y. Elovici, "Continuous verification using keystroke dynamics," *International Conference on Computational Intelligence and Security*, 2010.

[10] A. Acar, H. Aksu, A. Uluagac, and K. Akkaya, "Waca: Wearable-assisted continuous authentication," *IEEE Security and Privacy Workshops*, 2018.

[11] J.-S. Wu, W.-C. Lin, C.-T. Lin, and T.-E. Wei, "Smartphone continuous authentication based on keystroke and gesture profiling," *International Carnahan Conference on Security Technology*, 2015.

[12] G. Peng, G. Zhou, D. T. Nguyen, X. Qi, Q. Yang, and S. Wang, "Continuous authentication with touch behavioral biometrics and voice on wearable glasses," *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 3, 2017.

[13] R. Kumar, V. V. Phoha, and A. Serwadda, "Continuous authentication of smartphone users by fusing typing, swiping, and phone movement patterns," in *2016 IEEE* 8^{th} *International Conference on Biometrics Theory, Applications and Systems (BTAS)*. IEEE, 2016.

[14] Y.-H. Chuang, N.-W. Lo, C.-Y. Yang, and S.-W. Tang, "A lightweight continuous authentication protocol for the internet of things," *Sensors*, vol. 18, no. 4, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/4/1104

[15] C. Shen, Z. Cai, and X. Guan, "Continuous authentication for mouse dynamics: A pattern-growth approach," *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2012.

[16] K. O. Bailey, J. S. Okolica, and G. L. Peterson, "User identification and authentication using multi-modal behavioral biometrics," *Computers & Security*, vol. 43, no. Complete, 2014.

[17] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, 1979.