Group-authentication schema for tiny and constrained IoT devices using secret-sharing algorithm and hash functions

Tomasz Krokosz [0000-0003-1518-5848] and Jarogniew Rykowski [0000-0001-7944-6061]

Department of Information Technology
Poznań University of Economics and Business, Poland
tomasz.krokosz{jarogniew.rykowski}@ue.poznan.pl

Abstract. This paper presents a proposal for a new and efficient method of solving the problem of device identity verification. The solution is preliminarily designed for Internet of Things (IoT) devices that utilize Bluetooth Low Energy and broadcast transmission, particularly geolocation beacons. The main research objective was to achieve an appropriate level of security and trust while maintaining reasonable energy consumption and minimizing the computational cost of cryptographic operations. Due to the limited resources of typical IoT devices, to mention memory, processing power, and battery life, which are significantly smaller than those of general-purpose computing devices, implementing traditional or even lightweight verification mechanisms is often impractical. The article describes an alternative method of authentication based on secret sharing and hash functions. The proposed approach involves storing specific parameters in the device's non-volatile memory during installation. These parameters are subsequently used to authenticate broadcast transmissions, assuming that each transmitted message is unique. Signal recipients can evaluate the sender's authenticity through the assistance of a trusted third party. Participants can also verify their own authorization to use the received data (e.g., data broadcasted by a location beacon). The solution is designed to maintain backward compatibility with devices and applications that do not require verification. To demonstrate the feasibility of the proposed method, a prototype implementation was developed in the C programming language and evaluated on devices equipped with the popular ESP32-C6 microcontroller.

Keywords: Internet of Things, Bluetooth Low Energy, Authentication, Secret Sharing, Security.

1 Introduction

The Internet of Things (IoT) has significantly expanded the capabilities of everyday objects. Devices now not only receive, collect, and analyze data but also transmit it for further processing. For example, a thermometer can do more than just display the current temperature – it can send a request to the heating control system to adjust the room temperature, based on historical data and appropriate software. In this way, our

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

immediate surroundings, whether at home, at work, or in public spaces, form a network of interconnected devices capable of exchanging data to enhance the comfort and quality of everyday life. The IoT ecosystem also includes devices that function as an invisible "background" layer. These are typically cameras, various types of sensors (e.g., smoke or motion detectors), and similar devices. They operate in a transparent manner and are often designed to be hidden from direct view, with minimal maintenance requirements. This necessitates that such devices be small, energy-efficient, and inexpensive to operate, also reducing the need for frequent battery replacements. However, these physical constraints, such as limited internal space and small battery capacity, prevent the integration of high-performance components capable of executing more advanced data processing algorithms. These include cryptographic algorithms for securing data and transmissions, which are successfully implemented on more capable computing platforms such as PCs and mobile devices. In the context of constrained IoT environments, it is often not feasible to fully implement mechanisms that ensure fundamental security properties such as confidentiality, data integrity, and system safety. As a result, the overall level of security in such devices tends to be low [1], and this is further heightened by the fact that security is often not a primary design priority when compared to the desired functionality of the device. The limited availability of resources and layered architectures of most IoT applications force a trade-off: either functionality is prioritized, or security is compromised [6]. Due to business considerations [8] and activities such as updates [9], lack of standardization [7], and a general lack of user awareness regarding potential threats, functionality is typically given precedence.

There was a significant research performed recently to bypass the above limitations. Apart from discussions on more or less classical (i.e., already used with success for bigger computers) solutions to IoT domain, as well as such modern solutions as blockchain and machine learning [2, 3], some authors point out the heterogeneity of IoT applications and technologies used [4, 12], complex issues of attack types and security strategies [5], and many other aspects. Most researchers stated that there is a need for a new class of algorithms, better suited to the restrictions of the IoT domain. Their efforts resulted in creating a new domain of research – so-called lightweight cryptography [14]. In early research, the majority of the proposed algorithms were based on block/lattice ciphers and simple computations on small integer numbers [15], to operate with a few kB of RAM memory, with timings measured in tens of milliseconds, and 128-bit (or equivalent) keys. More recently, several authors concentrated on proposing new schemes for lightweight digital signing [10], also applying elliptic curves [13]. However, more efforts are needed to limit the computation timings and memory used [16], even if some tasks are shifted to upper layers (edge servers, fog, cloud, etc.) [17].

What a few years ago became of a particular interest was the fact of upcoming quantum computers. Thus, a need for new cryptography algorithms arose, resistant to quantum attacks, forming so called post-quantum cryptography [18]. Among others, hash functions were pointed out as an effective base for this new generation of algorithms. Due to small demands for CPU and memory, hash functions are well-suited for IoT [11]. Several techniques based on hash computations and Lamport-Merkle trees were proposed [23], including hash-based signatures for IoT [19], working with classical key

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

length and typical IoT devices [20], and even for the smallest 8-bit AVR processors [21]. This research direction seems to be highly valuable and promising.

It seems that, however, even the lightweight versions of algorithms are not suitable for the smallest IoT devices. As discussed above, since the implementation of cryptographic algorithms requires adequate hardware resources, deploying most cryptographic methods on IoT devices is highly challenging, if not entirely infeasible. The first major limitation lies in the devices' restricted memory capacity. Even if non-volatile flash memory is sufficient to store the program code, the small size of volatile RAM, often measured in mere hundreds of bytes (B), may be insufficient to execute multiple iterations of a cryptographic algorithm. For example, typical signature sizes for popular Falcon and Dilithium algorithms are rounded at the level of 2-4kB, which goes much beyond the operational memory of several popular small-size microcontrollers [10]. Even if we perform byte-by-byte computations followed by immediate transfer to a remote node, the heap size needed to achieve this goal bypasses memory capacity. Another critical constraint is the execution time of such procedures. In most cases, the time required to perform cryptographic operations would be unacceptable and could even prevent the device from fulfilling its primary function. Returning to data reported in [10], timings for the above-mentioned algorithms are in the range 100-800 ms for middle-sized microcontrollers such as ESP32. Comparing with, e.g., an activation period for a geolocation beacon – 350-400 µs in one-second intervals, we clearly see that none of these algorithms can be directly applied. Such long timings of an algorithm may impose a heavy load on system resources, including the power supply, leading to significantly faster battery depletion. For example, if we apply the fastest Dilithium algorithm (about 60 ms with ESP32) each time a beacon is activated, this action would enlarge the activity time by a factor of 200, and thus limit the battery life in the same manner (from a few years to a few weeks only). Some cryptographic algorithms require data initialization each time they are executed, which involves receiving specific parameters from the network beforehand. This process increases both bandwidth usage and energy consumption. Moreover, a significant subset of IoT devices operates exclusively in broadcast mode (such as the above-mentioned geolocation beacons), meaning they are not capable of receiving data at all. This makes most of the digital-signature algorithms, including the above-mentioned Falcon and Dilithium, practically unusable.

The above analysis highlights a critical need for new verification methods tailored to environments with severely constrained resources. Although classical techniques, widely adopted in the realm of high-performance computing, and their lightweight versions for IoT domain are highly effective, they are largely inapplicable to the smallest and most restricted devices due to the limitations outlined above.

The main objective of this article is to propose a novel method for authenticating devices that are highly constrained in terms of both computational resources and energy consumption, as well as for verifying users' rights to access services provided by such devices. To achieve this goal, Shamir's Secret Sharing (k, n) algorithm and hash functions were employed. The proposed solution is specifically designed for the smallest IoT devices. The target application domain includes commonly used devices such as geo-localization beacons, which operate using Bluetooth Low Energy (BLE) technology and the iBeacon protocol. An additional objective was to develop a proof-of-

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

concept implementation in the C programming language, which was tested on ESP32-series microcontrollers (ESP32-C6). The selected microcontroller variant features a built-in hardware module for computing hash functions (SHA), which enabled the preparation of two implementation variants: one leveraging the hardware accelerator for performance optimization and another relying solely on software. This dual implementation allowed for a comparative evaluation of the performance and efficiency of both hardware-assisted and software-based authentication routines.

The remainder of the paper is organized as follows. Chapter 2 describes the proposed solution, including the communication scheme and the procedures to be performed by each party. The third chapter details the authentication process for both the beacon and the client-beacon pair. Execution times of the proposed algorithm, based on the implementation results, along with corresponding graphs, are presented in Chapter 4. Finally, Chapter 5 concludes the article with a summary, key findings, and suggestions for future work that aim to expand the capabilities of the presented system.

2 Proposed Solution Overview

Our proposed solution assumes backward compatibility of beacon transmissions. This means that (1) all data broadcasted by the beacon are to be accepted by standard BLE scanners, and (2) recipients who do not wish to verify the beacon's signal can simply ignore the additional authentication data sent by the beacon. The solution relies on two types of messages transmitted by the beacon: the first is a classic message containing the UUID, MAJOR, and MINOR fields, which remain constant over time; the second is an authentication message, which includes verification data that changes over time. The authentication signal is generated based on specific information stored in the beacon's non-volatile memory at the time of installation (alongside the UUID, MAJOR, and MINOR values), as well as a random value that is different for each transmission. These messages are transmitted alternately, or the additional authentication signal is sent less frequently, for example, every n^{th} classic transmission, depending on system requirements and the beacon's broadcast interval. This signal is derived from a portion of a secret stored in the device's non-volatile memory and a previously mentioned random number. These two inputs are processed by a hash function, whose output constitutes the primary component of the supplementary data used to authenticate the beacon.

Any user within the transmission range of the beacon will receive the data broadcasted by this device (both signals). To ensure the reliability of the data, the user is required to initiate the verification procedure by forwarding the received information to a trusted third party, namely the Center Authentication (CA). This data may (but is not required to) include additional information regarding the user's own authentication. Upon receiving the necessary data, the CA performs the required operations and calculations (according to Adi Shamir's scheme [24]), and subsequently transmits the corresponding information back to the user. It is important to emphasize that, at no point during the communication process, does either party share any portion of the secret it holds with another party. Moreover, since each transmission incorporates a random factor, a "record and replay" attack is rendered infeasible.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

To assess the proposed solution, an implementation was developed in the C programming language and tested on widely used ESP32 series processors. The ESP32C6 variant was selected for the evaluation, as it is equipped with a dedicated hardware module for computing the hash function value (SHA). For comparative purposes, two versions of the code were created: one utilizing the aforementioned module to optimize computation time, and another without its use. The results of this evaluation are presented in the subsequent section of the chapter. The code is completely portable to any similar 32-bit platform, such as STM32 or nRF52. However, we did not provide a comparative research on timings for these families of microcontrollers – due to the simple computations involved, it is expected that the proportions of the timings are similar. The detailed comparison of these and other platforms is planned as future work.

2.1 Data flow

The consequence of one-way Bluetooth Low Energy (BLE) communication is the necessity of assigning a required set of parameters to the device in advance (during installation). For this purpose, the device's non-volatile flash memory was used to store all the necessary data. It was assumed that devices of the same type share a common secret and the same identifier. This means that all deployed beacons are assigned to one of the defined device categories, and for all of them, there is a shared (category-specific) parameter that serves as the device group identifier (group UUID). However, each beacon can have a different supplementary identifier (MAJOR, MINOR) and a different value for its portion of the secret. This makes it possible to determine in which broadcast transmission range the receiving device is located. Such functionality can also be provided by other means, for example, by tracking the device's MAC address. The beacon categorization described above ensures that in the client application, only one assignment to a device group is needed; each beacon is then treated the same way, and there is no need to store parameters for multiple devices separately. Of course, the group may consist of just one beacon, which ensures full backward compatibility with unauthenticated systems and traditional applications.

As mentioned above, the beacon broadcasts two messages. The first one contains data related to the beacon, i.e., the UUID, as well as the data required by the iBeacon protocol, namely the MAJOR and MINOR values. The second message contains data for authorization purposes, including, among others, a hash generated from the concatenated value of the secret portion assigned during installation and a value randomly generated at the time of transmission. The assigned portion of the secret is not disclosed at any stage of communication. The data transmitted by the devices is shown in Fig. 1.

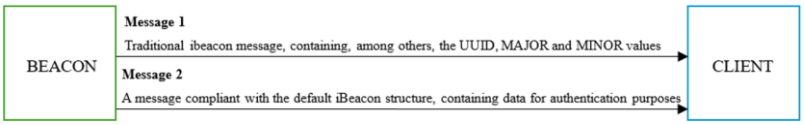

Fig. 1. Messages broadcasted by the beacon (BLE transmission).

The client, who is using a dedicated application and is within the transmission range of the beacon, receives two successive messages. As outlined in the previous section,

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

one of these messages contains data characterizing the beacon, while the other serves as the basis for performing the authentication operation. The application user also possesses a portion of the secret assigned to them during the installation phase. When the user receives a signal from a given beacon for the first time, or when they wish to initiate the verification procedure upon request, they transmit the necessary data to the CA. The transmitted data does not explicitly include the secret shares. The only explicit value is a specified value (generated by the beacon, varying with each transmission), which serves as the input parameter in the transformation method. Additionally, the transmitted data includes the hash function value, which is derived from the generated value and the user's share of the secret.

If the user is required to authenticate to the CA, they follow a similar procedure with respect to their private data. The first task performed by the client application is to randomly select a numerical value, execute a previously agreed-upon arithmetic operation (involving their share and the randomly selected value), and generate a hash by concatenating these values. Upon receiving the request, the CA queries its database to check whether a record exists for the pair of parameters: device group and user identifier. If the CA finds a matching record, it verifies whether the session is still active (i.e., whether enough time has passed since the last message was received to terminate the session) and, depending on the session's status, either performs verification or sends the client the stored result of the previously performed procedure. Suppose the CA does not find a record or determines that the session has expired for the given parameters. In that case, it selects its own share of the secret (as a participant in the procedure) and begins the necessary calculations. The described communication scheme between the client and the CA is illustrated in Fig. 2.

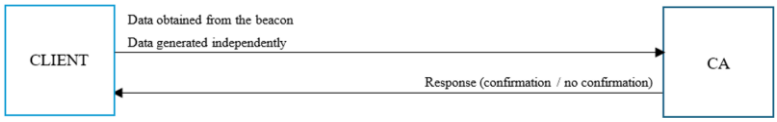

Fig. 2. Communication scheme between the client with a dedicated application and the CA (transmission over a wide-area network).

As mentioned earlier, both the beacon and the client do not send their shares in explicit form. Instead, they independently generate the previously mentioned value x and perform the agreed-upon operation (e.g., addition or multiplication) on their secret share with the randomly generated value. The randomly generated value is transmitted openly, along with the hash result of the pair of parameters: the secret share and the randomly generated value. In this way, the CA, upon receiving the data, has a complete set of necessary information for performing calculations. The CA has access to data from the installation parameter repository of all devices, so it can retrieve, among other things, installation data, i.e., the secret shares of the participants in the registration and the operation type. After retrieving the stored secret shares and the random value, it performs the arithmetic operation on its own and generates a hash. In the next step, it compares the received hash with the one it generated. If they differ, the data is deemed false, and the procedure is terminated – an appropriate response is sent to the user. However, if the values match, the CA also retrieves "its" own secret share and performs

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

a series of calculations to confirm the authenticity of the beacon and the client, as illustrated in equations 1-4.

$$l_0(x) = \frac{x-x_1}{x_0-x_1} * \frac{x-x_2}{x_0-x_2} \quad (1)$$

$$l_1(x) = \frac{x-x_0}{x_1-x_0} * \frac{x-x_2}{x_1-x_2} \quad (2)$$

$$l_2(x) = \frac{x-x_0}{x_2-x_0} * \frac{x-x_1}{x_2-x_1} \quad (3)$$

$$a(x)' = \sum_{j=0}^{2} y_j * l_j(x) = y_0 l_0(x) + y_1 l_1(x) + y_2 l_2(x) \quad (4)$$

The CA receives “masked” (i.e., different each time) secret values from group participants; therefore, it must perform computations to recover their explicit form. The values $l_0(x) – l_2(x)$ are used to carry out the polynomial interpolation procedure and, in the subsequent stage, to verify whether the resulting polynomial passes through the computed set of points. The difference lies in the fact that the operations are not performed on the explicit share value; instead, the beacon and the client perform the corresponding arithmetic operation on their own secret shares paired with the random value they generated. This allows for the "masking" of the private share, further increasing the level of security.

The CA, based solely on the "masked" secret value, independently performs the appropriate operations to obtain its explicit value (in this case, the reverse operation, i.e., subtraction). Based on this, the CA is able to obtain the original form of the function $a(x)$, which was defined and determined during the installation phase for this group of devices (in particular, for the pair). If the value of the function $a(x)$, stored in the repository, is identical to the one obtained through independent calculations based on the received parameters $a(x)'$, the verification procedure succeeds. Depending on the status, an appropriate response is sent to the user.

3 Implementation of the Authentication Process for the Beacon and the Client-Beacon Pair

The authentication technique described in the previous chapter exhibits some redundancy compared to the classical case when the beacon is not authenticated. The amount of data transmitted for beacon authentication is greater than that of a traditional, standard, and unauthenticated beacon message. Consequently, this scheme increases the demand on the transmission channel and, thus, the energy consumption by the beacon, shortening its operational lifetime. It should be noted that beacons are equipped with batteries that cannot be recharged. Once the battery is depleted, the device must be replaced with a new one, and the corresponding entries in the database storing active device identifiers must be updated. Therefore, certain simplifications were introduced that do not weaken the verification mechanism's strength. These changes primarily aim to reduce the amount of data transmitted by the beacon to the necessary minimum while maintaining the functionality of the proposed authentication mechanism. Additionally, it was assumed that not only the device (beacon) but also the service of allowing its use by a given client (from the application level that currently receives the beacon’s radio signal) can be a subject of the authentication. This is associated with two basic

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

authentication schemes: verification of the beacon signal regardless of the context in which the signal was received, and the one of the beacon-client pair. In the second case, the situation concerns the right of another device to use the localization service provided by that beacon at a given place and time. For example, the first scheme can be used to locate a product in a store, where each client "sees" all the products on the shelves. The second scheme is used for paying for the selected product and exiting the store with it. Only the client who has settled the purchase can open the exit gate.

The following fundamental modules were adopted in the solution.

1. Center Authentication (CA) – an independent entity responsible for the generation and verification of all authentication data.
2. Beacon – a localization device that, in traditional mode, emits its unique identifier locally, and in extended mode, broadcasts information used to authenticate that identifier; the beacon does not communicate over a wide-area network and cannot receive any transmissions.
3. Client – an application or device that receives the beacon's local radio signal and has wide-area network access to the CA, enabling verification.

It is assumed that all information transmitted by the beacon complies with the standard iBeacon format, meaning it contains 20 bytes (20B) of data payload (Fig. 3).

Field Name	iBeacon Prefix	UUID	Major numer	Minor numer	TX Power
Size	9B	16B	2B	2B	1B

Fig. 3. iBeacon format

The information transmitted by the beacon includes two iBeacon-format messages. In the first, traditional message, the beacon transmits its identifier UUID (16B) as well as the MAJOR and MINOR parameter values (each 2B in size). This information remains unchanged throughout the beacon's lifetime. This allows communication with the beacon without requiring its authentication, enabling the use of currently available software (BLE beacon scanners) without any modifications. In the second message, whose data content differs with each subsequent transmission, the beacon transmits information that can be used for its verification. This second message has the same length and format as the first, meaning it is entirely acceptable by existing software (BLE scanners). However, due to the unknown and variable nature of the data, it will be ignored by such software.

The authentication process is initiated by the CA and consists of two phases. In the first, installation phase, the CA generates a second-degree polynomial ($y = ax^2 + bx + c$) and stores its parameters a, b, and c. With the function definition available, the CA randomly selects a number x_n as the argument of the polynomial and calculates the corresponding function value y_n. This value is stored by the CA as its secret. The CA then calculates the function value for another argument x_m and this number is stored in the beacon's non-volatile memory (in double format, 8B). From that point on, the beacon is capable of generating an authentication signal, as described below.

The defined polynomial is shared by the entire group of beacons with the same purpose, for example, representing the beacon type, its owner, etc. However, a different argument value x_n is randomly selected for each beacon in the group, and a corresponding function value y_n is calculated. Each beacon only knows its assigned y_n. It does not

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

know the corresponding argument x_n, and the parameters required to determine the polynomial are also kept secret. If client authentication is also required, the above procedure is repeated, and each client is assigned a different y_n value, calculated from a randomly selected x_n. It is important to note that beacons, just like clients, have no knowledge of each other, nor of the values individually installed in other beacons. However, all devices that received installation parameters via the above procedure, parameters derived from the same polynomial, belong to the same group. The CA can verify this, as it knows the arguments assigned to both beacons and clients, and is capable of calculating their corresponding function values.

The verification phase begins when the client enters the beacon's transmission range and receives two messages from it. As already mentioned, in the first message, the beacon transmits its UUID identifier and the values of the MINOR and MAJOR fields. This message remains unchanged throughout the beacon's operational period. The second message is divided into two parts. The first part consists of two double-type numbers (each 8B long), encrypted using a hash-based message authentication code (HMAC). This data is stored in the field traditionally used for the UUID. The first number is a previously generated random value, converted to the double type. The second number is the x_n function value assigned during the installation phase, specific to that beacon. Both values are processed with the HMAC function to produce a 16B hash (Fig. 4). It is important to note that this second message changes with every transmission, since a new random number is generated each time, resulting in a different hash in each iteration. In the second part of the message, which is 4B in size, the beacon transmits the aforementioned random value (which changes each time), placed in the field corresponding to the combined MAJOR and MINOR fields.

Transmission n			
Field name	**UUID**	**Major Number**	**Minor Number**
Content	79c794728c6beb64	0a	90
Size	16B	2B	2B

Transmission n+1			
Field name	**UUID**	**Major Number**	**Minor Number**
Content	C19efcc1f5833281	4a	c8
Size	16B	2B	2B

Transmission n+2			
Field name	**UUID**	**Major Number**	**Minor Number**
Content	79c794728c6beb64	0a	90
Size	16B	2B	2B

Transmission n+3			
Field name	**UUID**	**Major Number**	**Minor Number**
Content	Bc1a4d6eaebda8ff	2d	3f
Size	16B	2B	2B

Fig. 4. Information payload of the additional message, including the sequence of consecutively transmitted messages.

After receiving the message, the client records three values: UUID + MAJOR + MINOR, HMAC(double, double), and the random value. These collected data are then sent to the CA. Upon receiving the data, CA uses the UUID to retrieve the function value assigned to the beacon during the installation phase from its database. It then independently computes a hash using the received random value and the retrieved function value. If this computed hash matches the data sent by the beacon, the CA sends an appropriate message to the client indicating the beacon's authentication status.

As mentioned earlier, the above scheme can be extended to include verification of the client's right to use a given beacon by leveraging secret sharing. For this purpose, the client sends not only the data received from the beacon to the CA, but also its own random value and a hash generated for the pair (random value, function value received from the CA). At this point, CA possesses three function values: one assigned to the beacon, one assigned to the client, and one stored for the given group. It also remembers the corresponding function arguments. Based on the system of three equations with

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

three unknowns (the polynomial coefficients: a, b, and c, i.e., the values in the equation $y = ax^2 + bx + c$, CA can solve for the unknowns, i.e., determine the polynomial's coefficients, following the method described in Equation 4. If the coefficients calculated in this way match those assigned to the group by the CA during the installation phase, the client is authorized to use the beacon because both the beacon and the client belong to the same group. This means the client has the right to access the beacon's signal to receive specific information or services, which can then be automatically offered to them. Otherwise, access to the service remains blocked, as the values received by the client from the beacon and forwarded to the CA prevent the CA from calculating the coefficients of the registered polynomial.

The above scheme can easily be extended to secret sharing by more than three entities. In the presented scenario, the group is formed by a single beacon, a client, and a CA. Even if the beacon and the client tried to reconstruct the secret based on the data they have, it would be impossible. To determine the form of the quadratic function, the minimum number of points required to calculate the function's coefficients is three. However, the group can consist of more than one beacon, or more clients might need to "cooperate" in order to use a single beacon. To achieve this, it is enough to generate a polynomial of degree n, where n is the number of devices cooperating within a single group (including CA). It is important to ensure that the number of shares is provided according to the threshold (the algorithm's parameter). For example, authentication for a pair of clients and their common beacon requires a fourth-degree polynomial.

It should be noted that solving the system of n equations described above does not involve raising to the power of n – only operations such as addition, subtraction, multiplication, and division are performed. Therefore, the computational complexity increases linearly, not exponentially, as the number of authenticated elements grows.

4 Implementation and Execution Timings

The computational and transmission overhead of a fully authenticated beacon, measured by its activity time and consequently the amount of energy required for performing the calculations and transmitting the additional message, is related to the necessity of transmitting two messages instead of one and performing the hash function computation. Assuming that the transmission time for a single iBeacon message is approximately 350÷400 µs (which results from the BLE transmission standard speed of 1 Mbps), and the time to compute the hash function value, and thus prepare the second message, is approximately 30 µs (using the hardware HMAC generator from the ESP32C6 processor) or about 180 µs (when calculating the HMAC function value in software), calculated total beacon activity time to be at least 400-600 µs, which means a 15-50% increase for a single transmission, and 115 – 150% for two succeeding transmissions (one unencrypted, for backward compatibility, and the second one encrypted for authentication purposes). A detailed analysis and assessment of the beacon's activity period repeatability is presented at the end of this section. Notably, the use of the hardware HMAC generator is especially interesting, as it only reduces the total activity period of the beacon approximately by half, from about 4-5 years to 2-3 years, which is

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

acceptable in most applications. It should be noted once again that using traditional authentication methods (e.g., digital signatures performed by specialized hardware units, or lightweight digital signing) shortens the beacon's activity time from several years to a few weeks, due to the very high energy consumption required for encryption.

The vulnerability analysis indicates that breaking the security using brute-force methods is complicated to the extent that the strength of the used hash function (in this example, HMAC with 128 bits) allows. To increase the security strength, the hash function can be extended to 256 bits (32B), and all calculations are performed on long-double numbers (16B for the ESP32C6 processors and the IDF environment used in the implementation). However, this would require the introduction of a third message from the beacon (which would halve the device's "lifetime"), or extending the second message by 16B, which would result in the abandonment of using standard BLE scanner software (while maintaining the same "lifetime" reduction). It seems that, for now, a 128-bit hash function is sufficient (at least for most geolocation applications).

Breaking security through methods other than brute force is impossible because the client (and, of course, the beacon) has practically no valid data for this purpose. The client does not know any value of the polynomial other than its own. It also does not know the argument (x), that was used to generate this value, meaning it cannot reconstruct the parameters of the polynomial. Neither the beacon nor the client reveals their explicit polynomial values ($a(x)$) in subsequent transmissions – the transmitted data is effectively hidden through the hash function for the random parameter. At the same time, converting the random number to a double type may involve some additional calculations using the function value y_n obtained during the installation phase, which further complicates breaking through a "brute-force" method. Specifically, this could again be a hash function like HMAC, but for example, for the sum of the random value and the function value registered during the installation phase.

There is a way further to strengthen the system's protection against "record and replay" attacks. To achieve this, pseudorandom values (rather than truly random ones) can be used for the parameters. In the example implementation, the rule can be introduced that with each subsequent transmission from the beacon, the pseudorandom parameter increases. The CA remembers the last received value and only allows verification attempts where the transmitted parameter value is greater than or equal to the last received value for the given UUID beacon identifier. In this way, "historical" transmissions are rejected, and only the most recently received message is considered valid, making it impossible for clients to reuse previously received data from the beacon. The possible range of pseudorandom parameter values (around 4 billion combinations) ensures that each transmission is unique for a time period several orders of magnitude longer than the beacon's "lifetime" on a single set of batteries, assuming one broadcast transmission per second (which is a standard in geolocation applications).

The schema presented above assumes that the beacon is not physically compromised (e.g., moved or cloned). To this end, we may store the secrets in an execution-only external module, such as a TPM (Trusted Platform Module) chip. However, using these chips would substantially increase execution time and, in turn, energy consumption. Note that even if a single beacon is compromised, this does not necessarily imply the compromise of the whole group or its clients – the other secrets remain safe.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

For the prepared program code, a total of 15,000 iterations were performed as part of the tests. Implementing hash function calculations using the hardware module (embedded in the ESP32C6 processor used for the test implementation) yields an average time of 35 µs per iteration. Slight and incidental aberrations from the average period were caused by interrupt handling and the need to activate the module from the code level. Based on the obtained data, the operation execution times were divided into intervals: up to 30 µs, 30-35 µs, 35-40 µs, 40-45 µs, and above this value. The results are presented in Fig. 5a. Slightly over 53% of the operations were executed within the 35-40 µs time range. Meanwhile, 6,065 results (the first two intervals) account for approximately 40% of the total results. There are 836 results with longer execution times, at around 45 µs, which constitute approximately 5.5% of all results.

In the case of performing the calculations solely in the program code (i.e., excluding the hardware HMAC computational unit, but using the same processor – ESP32C6), the average execution time was 180 µs. In this case, incidental extensions of this period due to interrupt handling were considerably more likely, resulting in a several-tens-of-µs increase in computation time. However, even in this case, the times required to obtain the HMAC value were stable and predictable. The obtained times were further divided into the following intervals: up to 170 µs, 170÷180 µs, 180÷190 µs, 190÷200 µs, and above 200 µs. Fig. 5b presents the visualization of the results for the adopted scenario. Almost 70% of all obtained times are shorter than 180 µs. Longer execution times are observed for 30% of the results, with over 4,500 results being shorter than 200 µs.

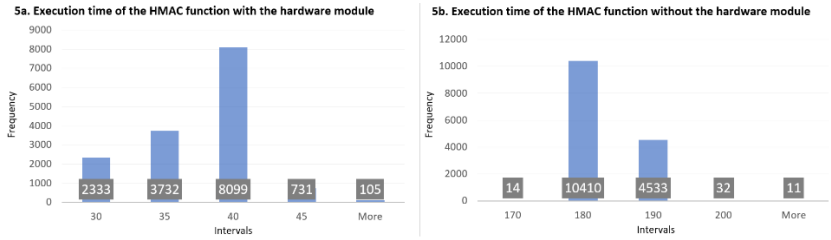

Fig. 5. Visualization of the execution times of the HMAC function a) calculations with the hardware module b) calculations in the program code, divided into intervals.

In the proposed solution, the BLE scanner software on the client side remains unchanged. This is due to the adopted data transmission organization. The first message represents the traditional, default form of communication in the iBeacon format, during which the beacon transmits, among other things, the UUID. The second, additional message, using the same packet structure, serves as the basis for performing the authentication operation. Sending the second message increases the transmission overhead by 100%, which is not insignificant in terms of energy consumption. As mentioned earlier, this will ultimately shorten the device's operational lifetime from approximately 5 years to 2-3 years, which is an acceptable value. However, since the CA is responsible for generating the data, mapping the values, and performing all verification operations, the implementation is feasible even with the least efficient microcontroller units. Additionally, since the procedure for preparing the verification data is not complex and consists of performing a set of simple arithmetic operations, the process itself is very fast, and the result is obtained in about 30 µs when the device has a hardware module, or about

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

200 µs when the code is executed from the program level. Such a minimal extension of the beacon's active period has practically no impact on its energy consumption and the operational time with a single battery set.

It is hard to compare the proposed approach with any other one dedicated to group authorization. Most of the approaches use classical (or lightweight, in the case of IoT) digital signature with private/public keys. The procedures needed cover bi-directional data exchange (which is not possible for the iBeacon standard) and serious computations. As mentioned in the Introduction, even the most effective, lightweight algorithms require tens (if not hundreds) of milliseconds to generate a signature, compared to the 35 µs in the case of the fastest implementation in our proposal. As we use a similar hash function (SHA), the cryptographic power of these solutions is similar. A more detailed analysis of our proposal for this parameter is planned for future work.

We decided to replace the analysis of the total energy used to perform the computations with the analysis of timings and time-to-live with a single battery. Due to the unusual way of activation of a beacon (a few hundred microseconds of activation, several hundred milliseconds of sleeping period), energy consumption is not linear. A beacon “rests” most of the time, which makes all the estimations of average energy consumption pretty useless. Thus, for this kind of IoT device, what is usually reported is the summary usage period, while powered by a single battery. As this period is counted in years (ranging for even the same type/model of a device from one to even 5-6 years, depending on the place of operation, environmental conditions – temperature, pressure, humidity, air flow, etc.), and it is hard to be theoretically estimated or simulated, it was not possible to really measure the overhead of our proposal in comparison with a non-authenticated beacon. However, by simply comparing the activation timings, we may say that the proposal cuts the lifetime of a device only by a factor of two, which is acceptable for most applications.

5 Conclusions and Future Work

The article proposes a new method of device authentication using Shamir’s Secret Sharing (k, n) algorithm and a hash function. The results of the conducted research indicate that the proposed solution is highly effective and feasible for deployment in environments with devices that lack efficient resources. The research work also included the implementation process. The prepared code, written in C, was executed on target Internet of Things (IoT) devices, for which the solution is primarily intended. The selected device was equipped with an ESP32C6 processor. Two scenarios were considered: with the hardware HMAC computational unit disabled and enabled. In the first case, the average execution time was 35 µs, while executing the operation from the program code took an average of 180 µs. These are delays that are imperceptible to the user and do not affect the comfort of using location services. Also, this does not result in an extended response time or cause communication delays. Since two messages are sent (one with data transmitted as part of the standard operation, and the second with authentication data), the transmission overhead increases by 100%. This, in turn, means

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

that the device's operational time, which is typically around 5 years, decreases to approximately 2-3 years. However, this is not an obstacle that would exclude the proposal from being implemented in production-level market solutions. A challenge was the proper data packing (required by the iBeacon protocol) to ensure that only 20B would carry the data necessary for a secure authentication process. Implementation demonstrated the practical feasibility of the system in resource-constrained environments.

Although it was not mentioned in the text, we should point out some possible directions for future research. For example, the level of risk should be estimated in case of CA malfunctioning, possible transmission errors, long-time observation of beacon transmissions, and their analysis towards similarities and repeatability/patterns, etc. What is worth researching is also using edge computing for CA functionality (local, or even ad-hoc group authentication). We also plan to investigate the power of lattice-based lightweight cryptography for generating digital signatures (such as already mentioned Dilithium and Falcon algorithms) with pseudo-random sequences to sign, more types of hash functions, optimized elliptic-curve cryptography (e.g., Ed25519, ECDSA), batch verification (SM2 algorithm and its successors), other schemes of threshold cryptography instead of Shamir's secret-sharing approach [22], and others.

The article serves as an introduction to practical solutions. First of all, even if the proposed solution is dedicated to geolocation beacons, it may be applied to any broadcast-only IoT device. Second, the authors see potential in this issue and are considering further research on authentication using the Karnin-Greene-Hellman scheme or another one, the Brickell scheme. The last-mentioned scheme is a generalized version of Shamir's scheme, and therefore, comparing the results between the two implementations could be the subject of future work related to the development and practical applications of the solution described in the article.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Neshenko, Nataliia & Bou-Harb, Elias & Crichigno, Jorge & Kaddoum, Georges & Ghani, Nasir. (2019). Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations. IEEE Communications Surveys & Tutorials. 21. 10.1109/COMST.2019.2910750.
2. Hassija, Vikas & Chamola, Vinay & Saxena, Vikas & Jain, Divyansh & Goyal, Pranav & Sikdar, Biplab. (2019). A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2924045.
3. Salah, Khaled & Khan, Minhaj. (2017). IoT Security: Review, Blockchain Solutions, and Open Challenges. Future Generation Computer Systems. 82. 10.1016/j.future.2017.11.022.
4. S. Sicari, A. Rizzardi, L.A. Grieco, A. Coen-Porisini, Security, privacy and trust in Internet of Things: The road ahead, Computer Networks, Volume 76, 2015, Pages 146-164, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2014.11.008.
5. Jayasree Sengupta, Sushmita Ruj, Sipra Das Bit, A Comprehensive Survey on Attacks, Security Issues and Blockchain Solutions for IoT and IIoT, Journal of Network and Computer

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23

Applications, Volume 149, 2020, ISSN 1084-8045, https://doi.org/10.1016/j.jnca.2019.102481.
6. F. Mehdipour, A Review of IoT Security Challenges and Solutions, 2020 8th International Japan-Africa Conference on Electronics, Communications, and Computations (JAC-ECC), Alexandria, Egypt, 2020, pp. 1-6, doi: 10.1109/JAC-ECC51597.2020.9355854.
7. Hameed, Ali & Alomary, Alauddin, Security Issues in IoT: A Survey. 1-5. 10.1109/3ICT.2019.8910320.
8. K. Kaushik and S. Dahiya, "Security and Privacy in IoT based E-Business and Retail," 2018 International Conference on System Modeling & Advancement in Research Trends (SMART), Moradabad, India, 2018, pp. 78-81, doi: 10.1109/SYSMART.2018.8746961.
9. Lata, Dr. Manju & Kumar, Vikas. (2023). Challenges to IoT Security: Industry Perspective.
10. Nielsen, Mads & Kjeldsen, Magnus & Turnip, Togu & Andersen, Birger. (2025). Post-Quantum Digital Signature Algorithms on IoT: Evaluating Performance on LoRa ESP32 Microcontroller. 592-600. 10.5220/0013508400003979.
11. J. Samandari and C. Gritti, "Post-Quantum Authentication and Integrity in 3-Layer IoT Architectures," 2024 21st Annual International Conference on Privacy, Security and Trust (PST), Sydney, Australia, 2024, pp. 1-11, doi: 10.1109/PST62714.2024.10788057.
12. Paez-Quinde, Maria & Guerrero, Javier & Alban, Robert & Narváez Rios, Magaly & Guachimboza, Marco. (2017). Cryptography Applied to the Internet of Things. 10.1007/978-3-319-73210-7_47.
13. Yavuz, Attila & Ozmen, Muslum Ozgur. (2019). Ultra Lightweight Multiple-time Digital Signature for the Internet of Things Devices. 10.48550/arXiv.1907.03911.
14. Rana, Muhammad & Mamun, Quazi & Islam, Md Rafiqul. (2021). Lightweight cryptography in IoT networks: A survey. Future Generation Computer Systems. 129. 10.1016/j.future.2021.11.011.
15. Patil, Abhijit & Bansod, Dr. Gaurav & Pisharoty, Narayan. (2015). Hybrid Lightweight and Robust Encryption Design for Security in IoT. International Journal of Security and Its Applications. 9. 85-98. 10.14257/ijsia.2015.9.12.10.
16. Nouma, Saif & Yavuz, Attila. (2023). Lightweight Digital Signatures for Internet of Things: Current and Post-Quantum Trends and Visions. 1-2. 10.1109/DSC61021.2023.10354177.
17. Nouma, Saif & Yavuz, Attila. (2024). Lightweight and Resilient Signatures for Cloud-Assisted Embedded IoT Systems. 10.48550/arXiv.2409.13937.
18. Bernstein, D.J. (2009). Introduction to post-quantum cryptography. In: Bernstein, D.J., Buchmann, J., Dahmen, E. (eds) Post-Quantum Cryptography. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-88702-7_1
19. HAN, Songshen & XU, Kaiyong & ZHU, Zhiqiang & GUO, Songhui & LIU, Haidong & LI, Zuohui. (2022). Hash-Based Signature for Flexibility Authentication of IoT Devices. Wuhan University Journal of Natural Sciences. 27. 1-10. 10.1051/wujns/2022271001.
20. Ghosh, S.K., Misoczki, R., & Sastry, M.R. (2019). Lightweight Post-Quantum-Secure Digital Signature Approach for IoT Motes. IACR Cryptol. ePrint Arch., 2019, 122.
21. Pereira, Geovandro & Puodzius, Cassius & Barreto, Paulo. (2016). Shorter Hash-Based Signatures. Journal of Systems and Software. 116. 10.1016/j.jss.2015.07.007.
22. Kim Kwang Choo, Debiao He, Lightweight distributed signature protocol for mobile computing and IoT devices, University of Texas System, https://patents.google.com/patent/US11310039B2/en
23. Alzubi, Jafar. (2021). Blockchain-based Lamport Merkle Digital Signature: Authentication tool in IoT healthcare. Computer Communications. 170. 10.1016/j.comcom.2021.02.002.
24. Shamir, Adi. "How to share a secret." Communications of the ACM 22.11 (1979): 612-613.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_23