Secure and Low-Power Authentication for Resource-Constrained Devices

Mohit Sethi[†*], Pranvera Kortoçi[*], Mario Di Francesco[*], Tuomas Aura[*]

[†]NomadicLab, Ericsson Research
[*]Aalto University, Espoo
{mohit.sethi,pranvera.kortoci,mario.di.francesco,tuomas.aura}@aalto.fi

***Abstract*—The Internet of Things (IoT) refers to an inter-connected world where physical devices seamlessly integrate into a global network and become active participants of business, information, and social processes. These physical devices are referred to as *smart objects* since they understand and react to the environment they reside in. However, deploying such Internet-connected smart objects is challenging, since they need to be correctly configured with appropriate network and security credentials. This is exacerbated by the fact that they have minimal input capabilities and may be in inaccessible locations. In this article, we describe how to employ the 3rd Generation Partnership Project (3GPP) Generic Bootstrapping Architecture (GBA) to ensure secure authentication and communication among a variety of devices and services. Although GBA relies on the infrastructure of mobile network operators, it requires no mobile network access but only IP connectivity to perform authentication. We show the feasibility of our approach with a prototype implementation that incurs in a minimal memory overhead. Experimental results also show that our solution is energy-efficient.**

Keywords*—*IoT, authentication, resource-constrained, AKA, GBA, energy-efficiency

I. Introduction

The Internet of Things (IoT) refers to an inter-connected world where physical devices are seamlessly integrated into the Internet and become active participants of business, information and social processes. The IoT relies on the interconnection of a large variety of heterogeneous devices and networks [1]. With technologies such as the Bluetooth low energy, ZigBee and embedded sensors, the physical objects in our vicinity have become objects that understand and react to the environment they reside in. These *smart objects* form the building blocks of the IoT [2, 3].

The Internet Protocol (IP) enables smart objects such as sensors and actuators to establish an end-to-end communication with other resource-constrained devices and services in heterogeneous networks. These smart objects are deployed in many applications that involve and require high security and integrity-awareness, such as security systems, industrial automation, health monitoring, and machine-to-machine communications [4]. The highly sensitive nature of the information being transmitted calls for robust, light-weight, and compact authentication as well as secure communication protocols that provide integrity and confidentiality. While all security protocols require these attributes, memory limitations, along with power supply restrictions of these devices make the problem of ensuring security more critical. Additionally, smart objects are often deployed in small spaces and inaccessible areas, which limits the possibility of regular maintenance for installing software updates and fixing security vulnerabilities. Moreover, deploying such ubiquitous smart objects is challenging because they need to be provided with appropriate network and security credentials despite their minimal input capabilities. In order to enable wide-scale deployment and acceptance of these smart objects, it is thus necessary to provide mechanisms for the initial configuration (i.e., *bootstrapping*) of network and security parameters without relying on human involvement.

One option for obtaining initial security credentials is to leverage an existing infrastructure. The security infrastructure of mobile network operators is an excellent candidate as it is scalable, dependable, and easy to use [5]. Such a security infrastructure has been standardized by the 3rd Generation Partnership Project (3GPP) under the Generic Authentication Architecture (GAA) [6]. One of its components, the Generic Bootstrapping Architecture (GBA), specifically targets automatic provisioning of shared keys between a device and an application server [7]. Specifically, GBA enables to bootstrap devices that have credentials in a Universal Integrated Circuit Card (UICC). Although GBA re-uses the credentials stored in the UICC, it requires no mobile network access but only IP connectivity to the infrastructure of the mobile network operator. Indeed, GBA could be a simple and usable bootstrapping mechanism for such resource-constrained devices as it does not require any human interaction. Accordingly, this article presents a secure and low-power implementation of GBA for constrained devices. In detail, we show the feasibility of using GBA even for devices with very limited resources in terms of memory and processing power. Our solution is based on an efficient implementation that incorporates only the bare minimum of the needed functionalities for secure authentication, without requiring any changes in the existing standards. We provide a prototype for an Arduino board and analyze its performance in terms of resource utilization. The obtained results show that our proposed solution incurs in a low power consumption.

The rest of the article is organized as follows. Section II overviews the relevant work in the literature. Section III introduces the reference security architecture and presents the design objectives of our work. Section IV presents some considerations for enabling GBA on constrained devices, followed by a prototype implementation. Section V introduces the testbed setup and presents the experimental results. Section VI discusses aspects related to security and outlines relevant use cases. Finally, Section VII provides some concluding remarks.

II. Related Work

Several solutions in the literature have tried to address the limitations of existing authentication protocols for IoT scenarios. For instance, Druml et al. [8] present a lightweight and flexible authentication method based on Elliptic-Curve Cryptography (ECC) that reduces the resource requirements at the devices. Specifically, the proposed solution moves the cryptographic processing from the resource-constrained device to an authentication terminal, thus mitigating the computational overhead at the device. Hummen et al. [9] also focus on an IP-based IoT scenario and introduce a delegation architecture which offloads part of the computation task to a delegation server. More precisely, they separate the connection establishment procedure from the actual protection of the application data with the Datagram Transport Layer Security (DTLS) protocol. Such an approach allows them to delegate the connection establishment procedure to an external server. As a consequence, the resource-constrained device no longer needs to implement public-key cryptography and can only leverage symmetric-key cryptography to secure application data. Hummen et al. [10] also claim that certificate-based DTLS is a suitable protocol for peer authentication in resource-constrained environments. The authors reduce the overhead generated in the DTLS handshake process to allow the protocol to be deployed in IoT scenarios. This reduction in the overhead is partially achieved via pre-validation of certificates, a session resumption method that minimizes transmission overheads, and (as already mentioned) delegating the association establishment procedure to more powerful devices.

The work by Raza et al. [11], similar to the one by Hummen et al. [10], aims at reducing the DTLS overhead. The authors apply header compression techniques while leveraging the IPv6 over Low power Wireless Personal Area Network (6LoWPAN) [12] standard. The authors show that their solution significantly reduces the amount of transmitted data while maintaining compliance with the DTLS standard. In contrast, we adopt an authentication approach that is assisted by the existing mobile operator network infrastructure and does not rely on certificates, handshake offloading or header compression.

Secure bootstrapping of ubiquitous devices has also been discussed in the literature. Sethi et al. [13] introduce a cloud-based system for bootstrapping ubiquitous displays that can also be generally applied to smart objects. The proposed solution relies on a pre-existing WiFi enterprise infrastructure and leverages active user participation to perform the initial configuration of the devices, which includes obtaining Internet access through a private wireless network. In contrast, a GBA-based solution does not require explicit user interaction for bootstrapping the devices. Furthermore, our approach does not rely on a specific access network such as WiFi.

Others in the research community have also investigated the suitability of GBA for IoT scenarios. For instance, Agarwal et al. [4] extend the GBA architecture to provide automatic authentication for machine-to-machine communication scenarios involving ZigBee-based wireless sensor nodes. While their solution exploits the existing cellular operator infrastructure to perform GBA authentication, they rely on specific features of the ZigBee specification. In contrast, our solution is not tied to a specific communication technology but only requires IP connectivity to the Internet. Han et al. [14] also adopt GBA to bootstrap security for a ZigBee-based wireless sensor network. Specifically, they rely on a mobile phone as a gateway between the ZigBee network and the operator infrastructure. They also introduce a new protocol for mutual authentication of the smartphone and the sensor nodes. On the contrary, we do not rely on additional network elements, such as gateways, to authenticate constrained devices. We rather allow the devices themselves to directly bootstrap security credentials by using the existing network infrastructure.

Finally, some considerations for using GBA in resource-constrained devices are also provided by Korhonen [15]. In detail, Korhonen proposes modifications to the bootstrapping procedure and advocates the use of CoAP instead of HTTP during the bootstrapping process to explicitly address the resource-constrained nature of IoT devices. In contrast, we show the feasibility of using a standards-compliant GBA authentication system for resource-constrained devices. We do not require any modifications to the existing infrastructure, but rather realize a careful implementation of the minimal subset of functionalities needed to implement GBA as it is currently defined.

III. Background

In this section, we overview the security infrastructure we rely on, followed by our design objectives.

A. The Generic Authentication and Bootstrapping Architectures

The Generic Authentication Architecture (GAA) [6] builds upon the existing cellular security infrastructure to provide general-purpose authentication for applications and services [5]. GAA specifies two authentication mechanisms. One relies on a public-key infrastructure and digital certificates, whereas the other one provides all communicating entities with a shared session key. The latter mechanism is called Generic Bootstrapping Architecture (GBA) [7], and constitutes a major building block of GAA. GBA leverages the cellular security infrastructure to *bootstrap* security credentials by using the permanent secret stored in the smart card of the User Equipment (UE), more precisely, in the Universal Integrated Circuit Card (UICC). Often, such a card is called the Subscriber Identity Module (SIM) card. GBA provisions both the UE and an application server with a shared session secret.

The Generic Bootstrapping Architecture (GBA) relies on the 3GPP Authenticating Key Agreement (AKA) protocol, which provides mutual authentication of the communicating entities and derivation of a session key to protect data exchange [6]. The combination of authentication and session key agreement constitutes the main idea of AKA. Besides, GBA leverages AKA to bootstrap a secure connection between a UE and an application server.

GBA consists of four major network elements: the Home Subscriber Server (HSS), the Bootstrapping Server Function (BSF), the Network Application Function (NAF), and the terminal or UE. The HSS contains subscriber data and manages their identity, access, and security. It shares a long-term master key K with the UICC in the terminal. The BSF is a GBA-specific server that facilitates the bootstrapping process between the UE and a target application server. In the GBA terminology, such an application server is called Network Application Function (NAF). In order to authenticate users

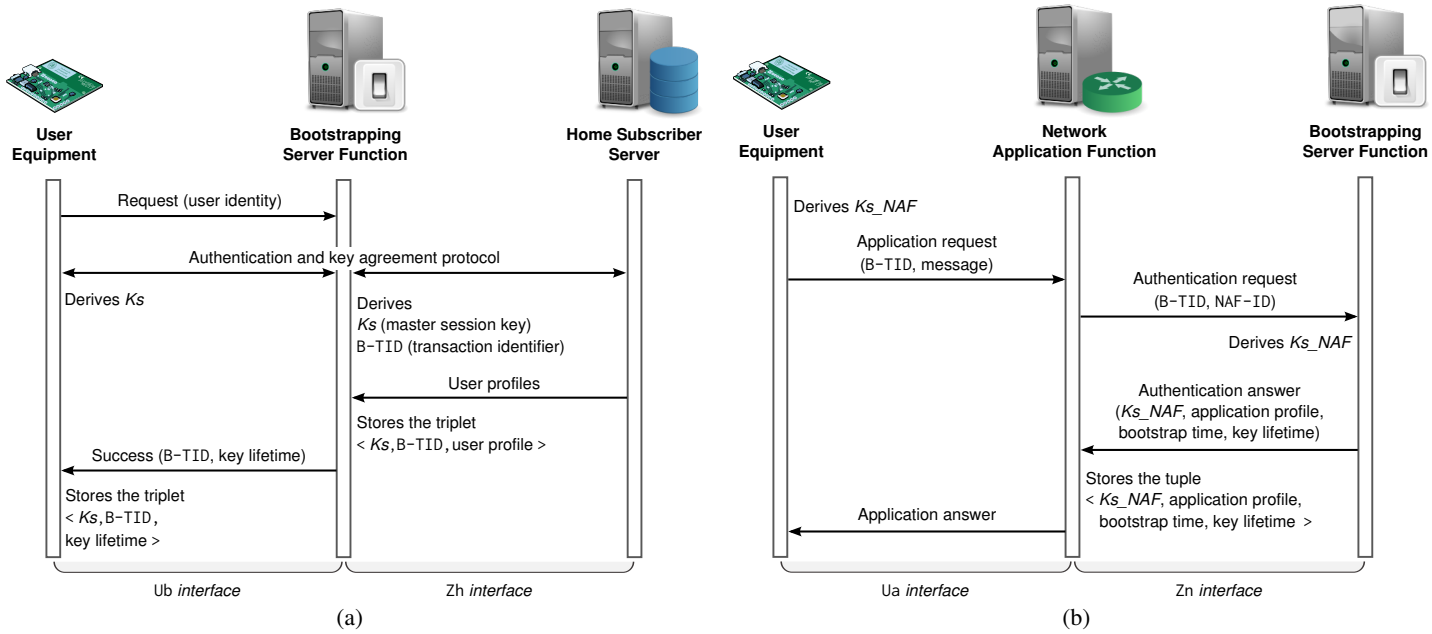

Fig. 1: GBA (a) bootstrapping and (b) usage procedures (adapted from [5])

with GBA, a NAF must have a secure communication link with the BSF. For instance, this secure link might be established with certificates and TLS. However, the 3GPP standard does not mandate any particular mechanism for establishing such a secure link.

The permanent secret shared between the UICC of the user and the mobile network operator is used to generate a time-limited GBA master key *Ks* that is shared between the UE and the BSF. The BSF then generates a NAF-specific session key *Ks_NAF* and distributes it to the appropriate NAF. The session key allows mutual authentication and protects the communication between the UE and the NAF. Figure 1a shows the bootstrapping message flow of GBA. The interactions between the network elements take place on top of *interfaces*. The bootstrapping procedure follows the message flow outlined below.

1. The UE initiates a bootstrapping procedure by identifying itself to the BSF with the identity known to the infrastructure of the mobile network operator. Such an interaction takes place over the Ub interface.
2. The authentication process between the UE and the HSS generates the session keys to be shared between the UE and the BSF, and a GAA master session key *Ks* is derived at the UE.
3. The BSF also derives the master session key *Ks* and constructs a Bootstrapping Transaction Identifier (B-TID), then receives user profile data from the HSS over the Zh interface. All these data are stored in the BSF database for the lifetime of the key.
4. The BSF sends then the session key *Ks* and B-TID along with the lifetime information to the UE.
5. Finally, the UE stores the data received in the previous step.

This message flow corresponds to the bootstrapping of the credentials between the UE and the BSF. The next step consists of actually using these credentials for authenticating the UE to the NAF.

Figure 1b shows how a shared secret is derived for mutual authentication and secure communication between a UE and an application server (NAF). Specifically, the bootstrapping usage procedure consists of the following message flow.

1. Whenever the UE attempts to access a given application in a NAF, it generates an application-specific session key *Ks_NAF* from the master session key *Ks*.
2. The UE sends a request to the NAF containing the B-TID along with the application-specific message. Such requests are sent over the Ua interface.
3. The NAF queries the BSF with the B-TID received in the previous step, along with its own identifier, the NAF-ID. Such messages are sent over the Zn interface.
4. The BSF then queries its database for the received B-TID. It also checks whether the NAF is authorized to receive application-specific keys for the given subscriber. The decision is based on the user profile information that the BSF has received from the HSS.
5. Once the BSF finds a matching B-TID in its database, it uses the corresponding *Ks* along with the NAF-ID to derive the application-specific key *Ks_NAF*.
6. Next, the BSF proceeds by sending the *Ks_NAF* key and the application-specific part of the user profile data to the NAF over the Zn interface.
7. Once the NAF receives and stores such data, the UE and the NAF can securely communicate by utilizing the *Ks_NAF* key to protect subsequent messages.

B. Design objectives

In designing our solution, we target the following functional objectives.

- **Suitability for low-end platforms**. The authentication process should be suitable for very constrained devices

built on low-end platforms, such as those including an 8-bit microcontroller and a few kilobytes of memory. The software should provide all needed capabilities itself whenever possible, at most relying on a few external libraries. The software should be compatible with different classes of constrained devices, in terms of both hardware and software features.

- **Access network transparency**. The authentication process should exploit the existing standard-compliant communication infrastructure. The designed solution should not require the constrained device to use a specific access network (e.g., cellular radio) but should support different communication technologies.
- **No user interaction**. The authentication process should be fully automated, i.e., it should not require any explicit user interaction with the constrained device as part of the initialization process. Conversely, the software implementation should not rely on the constrained device having input and output capabilities beyond those needed to fulfill network access.
- **Security**. The process should ensure the security of the authentication process involving the constrained device and a network service. It should also provide means to guarantee the authenticity, confidentiality and integrity of communications once a session is established.

IV. Implementation on constrained devices

In this section, we first provide some guidelines that enable implementing GBA on resource-constrained IoT devices. We then detail a prototype implementation of our solution for the Arduino platform.

A. Reducing resource utilization

Resource utilization can be reduced in the first place by simplifying the functionalities needed as much as possible, without compromising the compliance with the existing standards. A few related principles are detailed next.

HTTP stack requirements. According to the 3GPP specifications, GBA relies on either HTTP Digest [16], or Pre-Shared Key (PSK) TLS [17]. In the former case, the HTTP digest authentication requires the client to implement an HTTP stack. However, HTTP client libraries for resource-constrained devices are often unnecessary because the number of HTTP messages required for a complete GBA authentication transaction is very limited. Hence, using such libraries requires more memory than the actual communication-related data structures for message exchange. In order to avoid this memory overhead, the necessary messages can be hand-crafted and sent directly on top of TCP, without the need of a full HTTP stack. The plain text nature of the HTTP/1.1 protocol allows such an approach to work in practice. A sample HTTP request is shown below:

```
char httpReq[82] = "GET /naf/resource HTTP/1.1\r\n"
                   "Host: p123.example.net:8080\r\n"
                   "Connection: Keep-Alive\r\n"
                   "\r\n";
```

Similarly, the response can easily be parsed from the packets received from the transport layer, with no need to perform error checks. Receiving an unknown packet or data would lead to a re-start of the message exchange, thus saving computational resources for other purposes.

Cryptographic algorithms. Cryptographic algorithms are known to be computationally expensive. GBA relies on standard encryption algorithms, namely, the Advanced Encryption Standard (AES) [18]. Many resource-constrained platforms have an AES hardware engine, for instance, those shipping an IEEE 802.15.4 (Zigbee)-compliant radio module. This allows offloading the cryptographic computations to the AES hardware engine. Besides, optimized software-only implementations of AES suitable for IoT devices [19, 20] are also available for platforms lacking specialized hardware. In any case, it is possible to use AES in the reference IoT scenario we are considering in this work.

Memory requirements. Our solution copes with the resource constraints of IoT devices by purging unnecessary functionalities from memory when they are no longer needed. It is worth mentioning that, once the UE establishes a secure communication channel with the NAF, only the *Ks_NAF* and the B-TID need to be retained in memory. However, if the UE decides at a later moment to connect to other NAFs, the master key *Ks* and the data structures needed for its derivation need to be retained as well. Moreover, since GBA is used for initial authentication only, all the code and the libraries related to GBA can be purged from memory, thereby allowing the available memory to be reallocated for other application-specific purposes. Once the lifetime of the key expires, the GBA code and libraries can be loaded back into the memory. In scenarios employing session keys with a long lifetime, the device can even switch to other applications as well as routing or transport protocols (e.g., CoAP and RIPL/UDP) after secure bootstrapping. Such memory management enables GBA to be utilized for authentication and secure communication of devices with strict memory constraints.

Error handling. A typical GBA implementation requires a state machine to keep track of the messages exchanged with the BSF and the NAF. However, implementing such a state machine with appropriate error handling would consume a large amount of memory and computational power in IoT devices. Therefore, an easier and simple hard *failover* implementation is preferable for resource-constrained devices. For instance, once an erroneous or an unknown message is received, or a timeout occurs, the hard failover aborts the current GBA authentication transaction and starts over. Although such a solution reduces the complexity of the system and the related memory requirements, the energy consumption of the device may increase as a result of many failed GBA authentication attempts. Setting a limit to the number of failed attempts mitigates such a drawback. It is also recommended that devices enter a low-power sleep mode between failed authentication runs to save energy.

B. Prototype implementation

In this section, we describe our implementation of the GBA authentication specifically designed for constrained devices by following the principles outlined above.

For our prototype, we employed an Arduino Mega board and an Ethernet shield for interconnecting with the Internet. The Mega board[1] is equipped with an 8-bit ATmega2560 microprocessor, 8 KB of RAM, 4 KB of EEPROM and 256 KB of flash memory. We adopted Brian Gladman's byte-oriented

[1] http://www.arduino.cc/en/Main/ArduinoBoardMega2560

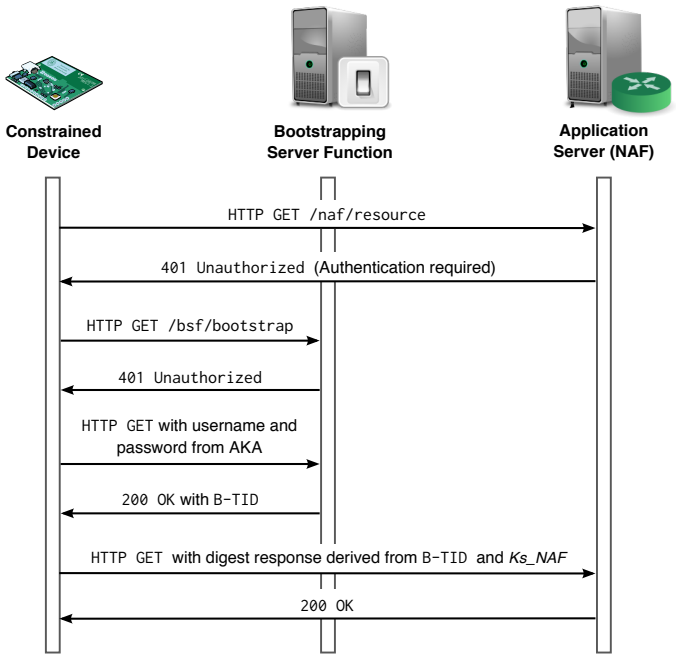

Fig. 2: Message sequence in the prototype implementation

AES implementation[2] for encryption, whereas we used readily available SHA256[3] and MD5[4] libraries for implementing the cryptographic hash functions. We wrote our software in the C programming language and provided minimal platform abstraction methods so that it can easily be executed on Linux-based embedded systems such as the Raspberry Pi. In our prototype implementation, we leveraged the default Ethernet and TCP libraries provided by the Arduino platform.

A representative sequence of the message flow in our prototype implementation is illustrated by Figure 2. The constrained device first contacts the NAF it intends to communicate with and sends an `HTTP GET` request to access the desired resource. The NAF responds to such a request with an `HTTP 401 Unauthorized` message with the `WWW-Authenticate` header set to indicate that digest authentication is needed for the device to access the given resource. The header of the HTTP contains the realm prefixed with `3GPP-Bootstrapping@`, suggesting the device to perform a GBA run in order to obtain the appropriate keys for digest authentication. As a consequence, the device contacts the BSF hosted by the serving network operator via an `HTTP GET` request. The BSF responds with a `401 Unauthorized` message with the HTTP header containing `algorithm=AKAv1-MD5`, thereby prompting the device to run an AKA authentication. The device then runs the AKA algorithm and sends another `HTTP GET` containing the username and password to the BSF. The BSF verifies the parameters and responds with a `200 OK` message, whose body contains the B-TID and indicates the lifetime of the B-TID and the derived master key *Ks*. Finally, the device contacts the NAF with an `HTTP GET` request containing the B-TID as the username and the *Ks_NAF* as the password. Here, *Ks_NAF* is derived from *Ks* and the fully qualified domain name of the NAF. The NAF verifies the supplied username and password, responding with a `200 OK` message if they are found to be valid. Following this message, the device and the NAF can communicate securely with the derived shared secret *Ks_NAF*. They can use this session key for TLS or DTLS in pre-shared key mode. Alternatively, this key can also be used to provide message-level encryption and/or integrity protection, for instance, in EAX mode [21].

[2] `https://github.com/abhay/gladman-aes`
[3] `https://github.com/Cathedrow/Cryptosuite`
[4] `https://github.com/tzikis/ArduinoMD5`

Parameter	Value
Program size	42,740 bytes
Data size	2,893 bytes
Average duration of one GBA transaction	1,473.6 $\pm$ 46.73 ms
Average power consumption	1,114.97 $\pm$ 6.55 mW

TABLE I: Resource utilization of the prototype implementation

V. Experimental Evaluation

In the following, we first briefly describe the testbed setup used for our experimental evaluation, together with the considered metrics. We then detail the obtained results.

A. Testbed setup and considered metrics

We set up our experimental testbed by connecting the Arduino Mega board to the Internet through the Aalto University network. We tested and evaluated our GBA implementation against standard 3GPP BSF and NAF interfaces running on Ericsson's public servers. In order to obtain information about the network traffic, we collected traces at the server by using the Wireshark network protocol analyzer [22], following the approach used by Lin et. al [23]. We utilized the Monsoon Power Monitor [24] to measure the power consumption of the constrained device. Specifically, we enabled the USB passthrough mode of the power meter and connected its USB port to the one on the Arduino board as power supply. We then connected the meter to a workstation that logged and processed the power consumption readings. In our evaluation, we replicated each experiment 15 times. We then derived the average values along with the related standard deviations.

In the experiments, we evaluated the following metrics.

- *Memory utilization*, as the amount of memory used by the prototype implementation on the considered platform.
- *Execution time*, as the time needed to complete one authentication transaction.
- *Average power consumption*, obtained by averaging the instantaneous power consumption reported by the power meter over the whole duration of an experiment.

The obtained results are detailed next.

B. Experimental results

The results about resource utilization of the prototype implementation are summarized by Table I. The first row indicates the overall size of the program, whereas the second row reports the size of the data segment of the executable. Such a parameter represents the size of the static data defined in the code. We can see how the size of the program and the data segment are both significantly below the maximum ones allowed by the hardware. It is worth noting that the data segment is stored in the main memory and does not include uninitialized global or static variables nor dynamically

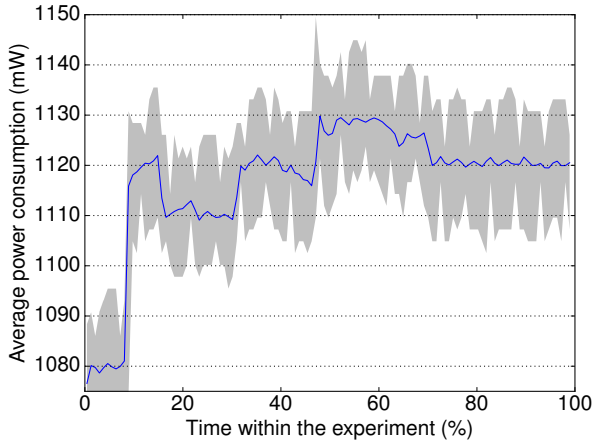

Fig. 3: Average power consumption as a function of time in a representative run of the authentication process

allocated data. However, the total memory usage is well below the 8 KB available in the Arduino Mega platform.

The third row of the table shows that the average duration of a single GBA transaction is equal to 1.47 s with a standard deviation of about 45 ms (i.e., 3% of the average). This includes the time from the very first connection until the final result is received, according to the message flow illustrated by Figure 2, and overall consisting of 8 messages. Such a duration is reasonable and overall rather limited, especially considering that the board needs roughly 10 s only to boot and initialize the network card.

Finally, the last row of the table shows that the average power consumption in the experiments is 1,115 mW with a very low variance (below 1%). Even though this value may appear rather high, it has to be considered with respect to the baseline power consumption, i.e., the one for an idle board. Therefore, it is better to consider the average power consumption as a function of time. Figure 3 indeed shows the corresponding measurements for a representative run of the authentication process. All samples logged by the power meter were aggregated with a granularity of 15 milliseconds. The figure shows the average value of the aggregated samples (i.e., the dark line) as well as the corresponding minimum and maximum values (i.e., the light gray bands). For convenience, the time is expressed as a percentage with respect to the total duration of the experiment (i.e., 1.75 seconds).

We can observe that the power consumption of the idle board is of nearly 1,080 mW. At about 10% of the experiment, there is a sharp increase in the power consumption. This happens as the device connects to the NAF. The power consumption first reduces, then increases again as the device is connecting to the BSF. The derivation of the secret material further increases the power consumption, that reaches about 1,130 mW after 50% of the experiment duration. The rest of the time includes the final part of the message exchange illustrated in Figure 2 that completes[5] the authentication process. Overall, the difference between the minimum and maximum power consumption is rather constant, roughly corresponding to 30 mW. Based on the obtained results, we can conclude that our software implementation incurs an overhead of about 45 mW with respect to the base power consumed by the idle board. As the average duration of a transaction is of about 1.5 s (Table I), the total energy overhead is of about 70 mJ. This result demonstrates how our implementation indeed achieves a low energy consumption.

[5]After establishing the session, the power consumption goes back to the initial value of 1,080 mW (not shown in the figure).

VI. Discussion

The security of our proposed solution relies on that of the 3GPP AKA. The latter has been thoroughly analyzed and (at least currently) does not present any serious security vulnerabilities as long as the network operator is trusted [25].

GBA is used for authenticating a device based on the 3GPP subscription credentials stored in the device. Normally the credentials are stored on a UICC, but also embedded UICCs (eUICC) are possible. eUICCs are targeted to scenarios where the 3GPP subscription should be changeable remotely, without having to physically replace the card in the device. Both UICCs and eUICCs provide hardware-based protection of the subscription credentials used for authentication in 3GPP networks. However, it is also essential to protect the access to the (e)UICC from applications to prevent man-in-the-middle attacks [26]. If a UICC or eUICC is not available, the GBA digest can be used [7]. The GBA digest uses Session Initiation Protocol (SIP) digest credentials, basically a username-password pair as well as a secret deducted from TLS, to perform GBA. The 3GPP requires that the SIP credentials are securely stored within the terminal.

Establishing a secure connection with an application server allows to perform remote configuration and management of the constrained device directly from the cloud where the application server resides [27]. To establish such a secure connection, the constrained device could be pre-configured with access credentials such as those provided by an UICC or eUICC. Unlike other methods, GBA-based authentication with pre-configured credentials does not require any interaction between the user and the device. The cloud service would then be used to configure and manage the device through the previously established secure Internet connection. This approach is particularly valuable for devices without (or with very limited) input capabilities such as those we are targeting in this study. Finally, the cloud-based solution may perform not only device management, but also data collection and visualization.

Our solution can be applied to several scenarios involving resource-constrained (possibly battery-powered) devices. Such scenarios include, for instance, cloud-based access control (e.g., with electronic locks) and remote home monitoring.

VII. Conclusion

In this article, we have shown the feasibility of using the Generic Bootstrapping Architecture (GBA) defined by the 3rd Generation Partnership Project (3GPP) to perform secure authentication of resource-constrained IoT devices. We have provided guidelines on how to efficiently realize a software implementation with low memory utilization by reducing the needed functionalities to the bare minimum, while interfacing with the infrastructure of mobile network operators without any changes in the existing standards. We have implemented our solution on an Arduino board and analyzed its performance in terms of resource utilization. Our experimental results have

shown that the device is able to securely authenticate with an application server in less than 1.5 seconds, thus resulting in a low energy consumption.

Acknowledgments

This work was partially supported by TEKES as part of the Internet of Things and the Cyber Trust programs of DIGILE (the Finnish Strategic Center for Science, Technology and Innovation in the field of ICT and digital business). The authors would like to thank Trung Hieu Nguyen for his help in analyzing the data collected during the experiments.

References

[1] M. Di Francesco, M. Raj, N. Li, and S. K. Das, "A storage infrastructure for heterogeneous and multimedia data in the internet of things," in *The 2012 IEEE International Conference on Internet of Things (iThings 2012)*, November 2012, pp. 26–33.

[2] M. Sethi, "Security in smart object networks," Master's thesis, KTH, School of Information and Communication Technology (ICT), 2012.

[3] M. Sethi, J. Arkko, and A. Keranen, "End-to-end security for sleepy smart object networks," in *The 7th IEEE Workshop on Practical Issues In Building Sensor Network Applications (SenseApp 2012)*, 2012, pp. 964–972.

[4] S. Agarwal, C. Peylo, R. Borgaonkar, and J.-P. Seifert, "Operator-based over-the-air M2M wireless sensor network security," in *The 14th International Conference on Intelligence in Next Generation Networks (ICIN 2010)*, 2010, pp. 1–5.

[5] D. S. Holtmanns, V. Niemi, P. Ginzboorg, P. Laitinen, and P. N. Asokan, *Cellular Authentication for Mobile and Internet Services*, 1st ed. Wiley Publishing, 2008.

[6] 3GPP, Technical Specification Group Services and System Aspects, "Generic Authentication Architecture (GAA)," http://www.3gpp.org/DynaReport/33919.htm.

[7] ——, "Generic Bootstrapping Architecture (GBA)," http://www.3gpp.org/DynaReport/33220.htm.

[8] N. Druml, M. Menghin, A. Kuleta, C. Steger, R. Weiss, H. Bock, and J. Haid, "A flexible and lightweight ecc-based authentication solution for resource constrained systems," in *The 17th Euromicro Conference on Digital System Design (DSD 2014)*, 2014, pp. 372–378.

[9] R. Hummen, H. Shafagh, S. Raza, T. Voig, and K. Wehrle, "Delegation-based authentication and authorization for the IP-based Internet of Things," in *The 11th IEEE International Conference on Sensing, Communication, and Networking (SECON 2014)*, 2014, pp. 284–292.

[10] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle, "Towards viable certificate-based authentication for the Internet of Things," in *The 2nd ACM Workshop on Hot Topics on Wireless Network Security and Privacy*, ser. HotWiSec '13, 2013, pp. 37–42.

[11] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight Secure CoAP for the Internet of Things," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3711–3720, 2013.

[12] Internet Engineering Task Force (IETF), "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," https://tools.ietf.org/html/rfc6282, September 2011.

[13] M. Sethi, E. Oat, M. Di Francesco, and T. Aura, "Secure bootstrapping of cloud-managed ubiquitous displays," in *The 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2014)*, September 2014, pp. 739–750.

[14] K. Han, J. Kim, K. Kim, and T. Shon, "Efficient sensor node authentication via 3GPP mobile communication networks," in *The 17th ACM Conference on Computer and Communications Security*, ser. CCS '10, 2010, pp. 687–689.

[15] J. Korhonen, "Applying generic bootstrapping architecture for use with constrained devices," in *Workshop on Smart Object Security*, 2012.

[16] "HTTP Authentication: Basic and Digest Access Authentication," http://tools.ietf.org/html/rfc2617, June 1999.

[17] "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)," https://tools.ietf.org/html/rfc4279, December 2005.

[18] "Announcing the Advanced Encryption Standard (AES)," http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, November 2001.

[19] J.-P. Kaps and B. Sunar, "Energy comparison of AES and SHA-1 for ubiquitous computing," in *Emerging directions in embedded and ubiquitous computing*. Springer, 2006, pp. 372–381.

[20] S. Didla, A. Ault, and S. Bagchi, "Optimizing AES for embedded devices and wireless sensor networks," in *The 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*, 2008, p. 4.

[21] M. Bellare, P. Rogaway, and D. Wagner, "The EAX mode of operation," in *Fast Software Encryption*, 2004, pp. 389–407.

[22] A. Orebaugh, G. Ramirez, J. Burke, and L. Pesce, *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Syngress Publishing, 2007.

[23] Y. Lin, T. Kämäräinen, M. Di Francesco, and A. Ylä-Jääski, "Performance evaluation of remote display access for mobile cloud computing," *Computer Communications*, to appear.

[24] Monsoon Power Solutions Inc., "Mobile Device Power Monitor Manual – Version 1.8," http://msoon.github.com/powermonitor/PowerTool/doc/Power%20Monitor%20Manual.pdf, June 2012.

[25] M. Zhang and Y. Fang, "Security analysis and enhancements of 3GPP authentication and key agreement protocol," *IEEE Transactions on Wireless Communications*, vol. 4, no. 2, pp. 734–742, March 2005.

[26] 3GPP, Technical Specification Group Services and System Aspects, "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Recommendations for Trusted Open Platforms," http://www.3gpp.org/DynaReport/33905.htm.

[27] M. Sethi, M. Lijding, M. Di Francesco, and T. Aura, "Flexible management of cloud-connected digital signage," in *The 12th International Conference on Ubiquitous Intelligence and Computing (UIC 2015)*, August 2015.