Symmetric and Asymmetric Schemes for Lightweight Secure Communication

Simona Buchovecká(✉), Róbert Lórencz, Jiří Buček, and Filip Kodýtek

Department of Information Security, Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic
{simona.buchovecka,lorencz,jiri.bucek,kodytfil}@fit.cvut.cz

Abstract. The paper deals with the topic of lightweight authentication and secure communication for constrained hardware devices such as IoT or embedded devices. In the paper, protocols based on both symmetric and asymmetric schemes are presented, utilizing a PUF/TRNG combined module, showing it is advantageous to have single module that will allow generation of both TRNG and PUF at the same time. This approach minimizes implementation requirements and operational resource consumption. Moreover, it allows the simplification of the overall key management process as the proposed protocols do not require to store secrets on the devices themselves. This paper is the extended and revised version of the paper entitled "Lightweight Authentication and Secure Communication Suitable for IoT Devices" [1] presented at the 6th International Conference on Information Systems Security and Privacy (ICISSP) 2020.

Keywords: Authentication · Secure communication · PUF · TRNG · Key generation · Key management · IoT security

1 Introduction

With the rising usage of smart devices interconnected in Internet of Things (IoT) the importance of their security is growing. The primary security functions that needs to be established are secure authentication for ensuring only properly authenticated devices are connected to the system or network and secure communication to ensure the confidentiality of transferred data. As necessary prerequisite for both there is a need for proper use of cryptography, especially cryptographic key management, all with the constraint of limited computing resources and low power consumption.

Different communication protocols were proposed for secure authentication and communication in IoT world and are being implemented nowadays. These include machine-to-machine/Internet of Things connectivity protocol (MQTT), Constrained Application Protocol (CoAP), or Datagram Transport Layer Security (DTLS) that can be integrated with CoAP. However, there is ongoing development and lighter variants of the protocols are being continuously introduced – such as Lithe [36] or E-Lithe [20] as a lightweight variant of DTLS [46], because the originals are quite resource exhaustive for simple constrained devices.

S. Furnell et al. (Eds.): ICISSP 2020, CCIS 1545, pp. 97–114, 2022.
https://doi.org/10.1007/978-3-030-94900-6_5

Another problem that is often being neglected is the need for cryptographic key lifecycle and its management including the secure key generation, distribution and usage. The generation of cryptographic keys is the first and essential step in the key life cycle. The generated key needs to meet the strict requirement of its unpredictability, arising from Kerckhoffs' principle formulated by Auguste Kerckhoffs in 1883 [22]. The principle states that the cryptographic system should be secure even if everything about the system, except the key, is public knowledge. This principle is applied to all modern encryption cryptosystems and the algorithms for encryption are publicly known. Therefore, the key needs to be kept secret and unpredictable, so the attacker cannot easily guess it. In hardware, the Random Number Generators (RNGs) or Physical Unclonable Functions are used to generate unpredictable bitstream. Further, postprocessing of this bitstream allows to generate the cryptographic key.

An RNG can be defined as a device or algorithm which outputs a sequence of random (thus independent and uniformly distributed) numbers. In practical hardware implementations, the output sequence is represented as a bit stream of zeros and ones, that may be further sliced and converted to the integers, as per the need of the implemented algorithms. Nowadays, the RNGs are most often and most widely used for cryptographic key generation. The RNG can utilize non-deterministic effects in analogue or digital circuits such as noise generated in the circuit itself, including thermal noise, shot noise or avalanche noise or in case of programmable devices the advantage can be taken from the metastability of logic circuits. This is the resource and power efficient way of generating random bitstream. Once generated, keys should be stored in secure manner [19], to be protected against attacker. According to sensitivity and criticality of the information, various approaches for storage keys are used today in practical applications. For most critical applications Hardware Security Modules (HSMs) are being used, however, implementing the HSM function often required much more resources, than available on the constrained device. Therefore, the properly defined and implemented key management, consistent way of handling variety of cryptographic keys, including proper key generation, key storage, key usage for various applications (authentication, access control, encryption) and possibilities of reusing traditional security mechanisms and ensuring end-to-end integrity verification mechanisms in interconnected IoT and embedded systems, as depicted in Fig. 1 is still a challenging task [32, 37, 41]. The need for proper key management in particular applications of embedded systems and IoT started to be raised in some research papers with regards to specific areas such as automotive context [40] distributed sensor networks [8], or embedded systems in general [42].

Nowadays, PUF usage is promising to solve the issue of secure storage of cryptographic keys. The concept of PUF was originally introduced in [35] showing that instead of relying on number theory, the mesoscopic physics of coherent transport through a disordered medium can be used to allocate and authenticate unique identifiers by physically reducing the medium's microstructure to a fixed-length string of binary digits. Instead of storing the key in memory, the key is generated at the time it is needed. Moreover, PUFs are on-way, inexpensive to fabricate, prohibitively difficult to duplicate, admit no compact mathematical representation, and are intrinsically tamper-resistant, making them the ideal candidate for providing tamper resistant design for cryptographic key generation and storage.

This radically new approach to secure key storage utilizing PUF was defined in [19]. With regards to drawbacks of non-volatile storage, authors define the criteria for new approach: key should not be permanently stored in digital form on the device, key should be extracted from the device only when required, and after having been used, it should be removed from all internal registers, memories, and locations and key should be somehow uniquely linked to a given device such that it cannot be reproduced or the device with a same key manufactured.

Therefore, PUFs usage is promising to solve the issue of secure storage of cryptographic keys. Instead of storing the key in memory, the key is generated at the time it is needed. A combined PUF/TRNG circuit used in our paper is therefore a suitable alternative for the purpose of key generation and authentication in lightweight cryptographic applications, such as IoT devices and other embedded platforms.

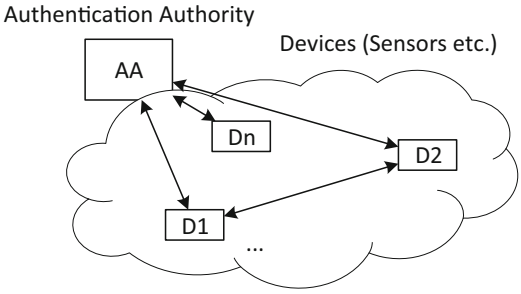

Fig. 1. Interconnected systems with an authentication authority [1].

The aim of this paper is to discuss protocols for authentication and secure communication utilizing PUF and TRNG, showing it is advantageous to have single module that will allow generation of both TRNG and PUF at the same time, since it minimizes implementation requirements and operational resource consumption. The goal we want to achieve in order to simplify the key management on the simple hardware devices and microcontrollers is to remove the requirent of storing secrets on the device itself. The paper is the extended and revised version of the paper entitled "Lightweight Authentication and Secure Communication Suitable for IoT Devices" [1] presented at the 6th International Conference on Information Systems Security and Privacy (ICISSP) 2020. This extended paper introduces the new schemes based on asymmetric cryptography (Algorithms 4 and 5). This brings the benefit that no shared keys need to be transmitted over secure channel. Further, the paper extends original Case Study, with the discussion on the length of the generated bitstream after all necessary corrections and experiment proving the quality of the generated key material.

This paper is organised as follows, related work and current State-of-the-Art is summarized in Sect. 2, providing us with the theoretical basis for our further work. In Sect. 3 our proposed approach to lightweight authentication and secure communication is presented, introducing the authentication algorithms based on symmetric and asymmetric cryptographic schemes, as well as secure communication approach. Section 4 presents

a case study and feasibility review of proposed protocols with a specific PUF/TRNG circuit. Section 5 concludes this paper.

2 State-of-the-Art and Technical Background

As stated in previous paragraphs, every protocol for authentication and secure communication relies on security of used secret cryptographic keys and the security of the cryptographic system is exclusively linked to the security of the key, as discussed for instance in [14]. In critical applications, especially when used in an untrusted environment, cryptographic keys should never be generated outside the system and they should never leave the system in clear. Therefore, if the security system is implemented in a single chip (cryptographic system- on-chip), the keys should be generated inside the same chip.

The minimum common requirements for key generation and storage are summarized by Maes et al. [30]. Every system implementing a cryptographic algorithm needs a source of true randomness that ensures unpredictable and unique fresh keys and a protected memory that will shield the keys from unauthorized parties and will allow the reliable key storage and usage. This implies the need for the true randomness source that will enable generation of random bitstream. The traditional methods of generating cryptographic keys in hardware and embedded systems are mainly based on true random number generators (TRNGs). As stated by Schindler [39], ideal random number generators are characterized by the property that the generated random numbers are independent and uniformly distributed on a finite range. Various TRNG designs suitable for cryptographic key generation include purely digital designs [12, 13], Phase-Locked Loops in designs targeting FPGAs [9, 15], Random access memories [17] or multiple designs [5, 16, 27, 45] based on Ring Oscillators as a source of entropy. However, implementation of mere TRNG is not enough to resolve the problem of secure key storage.

In [30] the idea of using PUF-based key generators is presented, as using PUF it is possible to fulfill both requirements on secure key generation and storage at once. PUF is a system responding to a challenge C with a response R, referenced in general as a challenge-response pair. According to [29] PUF is best described as an expression of an inherent and unclonable instance-specific feature of a physical object and as such has a strong resemblance to biometric features of human beings, line fingerprints, and thus cloning a PUF is extremely hard if not impossible at all. Therefore, there is no need for a protected non-volatile memory since the randomness is measured only when needed. However, the PUF output may slightly vary in different measurements, and it is still challenging to get static PUF output as required by cryptographic schemes.

Existing PUF designs proposed for cryptographic applications include PUFKY based on a ring oscillator PUF [30] providing low-failure rate, generation of read-once keys [23] single-chip secure processor for embedded systems [44], arbiter PUF for device authentication and secret key generation [43] and others.

Our goal is to enable a secure communication and authentication method using a combined PUF/TRNG circuit that will allow generation of keying material using both PUF and TRNG at the same time, utilizing benefits of each one. Session keys should be periodically re-generated and are not stored in non-volatile memory in a long term,

and as such the TRNG is ideal for this case. On the other hand, utilizing the PUF for asymmetric keys looks promising - as the decryption key should remain private it is ideal to utilize the PUF for key generation and storage, as the key remains private, never leaves the device and cannot be copied nor cloned to another device.

There have been several similar works published recently. The very first attempts in using PUF for the device authentication were rather simple. In [43] simple authentication against authentication authority was discussed, using pre-generated challenge-response pairs stored centrally. At the authentication time, the challenge is sent to the device and response then compared with the output. Same challenge cannot be reused again due to possible replay attacks, limiting the number of possible authentications of the device. More sophisticated PUF-based authentication protocols were reviewed in [10]. The work of [34] using reprogrammable non-volatile memory; Hammouri et al. [18] using two arbiter PUFs; protocol based on logically reconfigurable PUFs [21] which allows to recycle the challenge tokens; Reverse Fuzzy Extractor [47] allowing mutual authentication; Slender PUF protocol [31] that does not expose the full PUF responses, only the random subset instead; and Converse authentication Protocol [24] which provides one-way authentication of the server.

The protocols discussed above do not consider the need of key establishment, which is prerequisite in all the cases. Neither, the secure communication protecting the confidentiality of transmitted data is taking into the consideration. The [10], further discuss other caveats of the PUFs - responses being not perfectly reproducible, small output space of strong PUFs or need of secure TRNG, that is substantial for most of the protocols. Another issue is the privacy preservation of the devices being authenticated [2, 4] – as the PUF responses are unique per device and cannot be deliberately altered once device is manufactured, it is necessary to design the protocol in the way it preserves the privacy of the device.

3 Proposed TRNG/PUF Module for Secure Authentication and Communication

Our design aims to simplify the key management on the endpoint embedded device, allowing the efficient and secure authentication and communication, both against the Authentication Authority, as well as mutually across the interconnected devices. As discussed in the previous sections, we aim to utilize the single circuit for key generation using PUF and TRNG as a basic building block of the module so that there is no need to store secrets on the hardware device.

The overall module as presented in [1] is depicted in Fig. 2. It provides PUF authentication and PUF/TRNG based key generation. For the authentication, the PUF is used, since it provides randomness intrinsically present in the device and utilizes the fact that the generated response is unique per device. Since there is a need for both static key, as well as ephemeral keys, combination of PUF and TRNG is used in this case – the PUF is used for generation of static (private) key that never leaves the device, thus utilizing all the advantages of PUF, while TRNG is used for generation of ephemeral, one-time keys, that are shared with other communicating parties.

Asymmetric schemes are suitable if the private key is easily generated from random sequence by a Key Derivation Function (KDF), such as PBKDF2 [38]. For example, ElGamal encryption [11] and DSA/ECDSA signature schemes can be used, if good quality public parameters are chosen (generation of the public key from a private key is denoted as GENPK in Fig. 2). On the contrary, an RSA key requires more complex processing including secure prime generation. TRNG output is also used to generate random nonce and padding data.

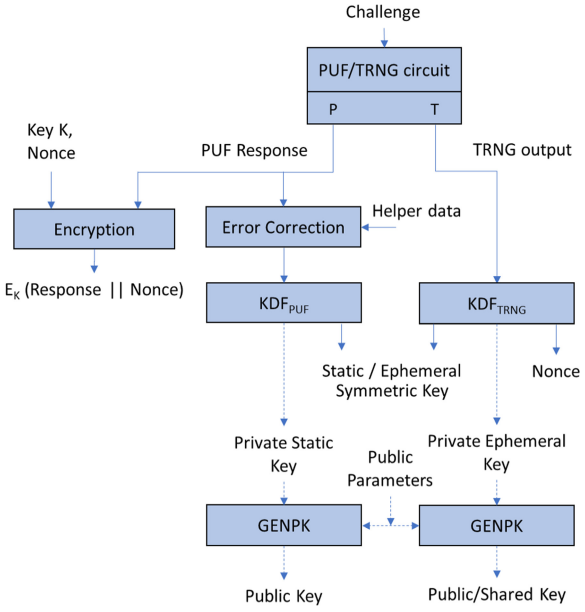

Fig. 2. Embedded module for secure authentication and communication. KDF is a Key Derivation Function, GENPK generates a public key from a private key and public parameters. Error Correction is used to obtain a stable key material from the PUF Response (see Sect. 3.1) [1].

3.1 Authentication Against Central Authentication Authority

Before the device is connected to the network and is allowed to communicate it must be properly authenticated. Since the PUF responses are unique per each device, and are intrinsically random, it makes PUF the ideal cryptographic primitive for device authentication. The authentication protocol against Authentication Authority using pre-generated challenge-response pairs that can be easily implemented in hardware devices is quite straightforward. This protocol does not require the PUF to have a large space of challenge-response pairs (even one challenge-response pair is enough). The authentication protocol consists of two phases – secure enrollment phase and authentication phase itself and is depicted in Algorithm 1 [1].

The Enrollment phase is critical for the security of all protocols based on PUFs, and (analogous to biometric authentication methods) must be performed in a secure

environment. We assume that a suitable environment can be for example during the manufacturing process, but specific means are not elaborated in this paper.

During the Enrollment phase of Algorithm 1, the challenge/response pair(s) (C, R) are measured from the targeted device and securely stored at the central authenticating authority (AA), that can be either integrated into the gateway or be represented by separate device that the gateway is querying during authentication process. A database DB_{Di} of the pairs (C, R) is created for each device D_i. Furthermore, the public key (PK_{AA}) of the authenticating authority is pre-set on the device, so as the authentication data can be securely transferred. For this purpose, an asymmetric scheme (ElGamal) can be used, as proposed in the section above. We assume that PK_{AA} is protected against unauthorized changes (by the tamper-evidence property of the PUF).

The first 4 steps of the Enrollment phase are common for all 3 algorithms presented in this paper. The database DB_{Di} is used also in the Authentication phases of Algorithms 2–3.

Algorithm 1: Authentication against central Authentication Authority [1].

Enrollment phase (*Secure environment*)
Common for Algorithms 1 – 3:
1. **AA** → **D1**: Challenges (C_1, C_2, ...)
2. **D1**: $R_1 = PUF(C_1)$, $R_2 = PUF(C_2)$ ...
3. **D1** → **AA**: Responses (R_1, R_2, ...)
4. **AA**: Store (C_i, R_i) to DB_{D1}

Specific only for Algorithm 1:
5. **AA** → **D1**: Public key PK_{AA}
6. **D1**: Store(PK_{AA})

Authentication phase for D1
1. **AA**: Choose (C, R) from DB_{D1}
2. **AA** → **D1**: Challenge C, Nonce N
3. **D1**: R' = PUF(C)
4. **D1** → **AA**: CR = $E_{PK_{AA}}$(R' || N)
5. **AA**: (R', N') = $D_{SK_{AA}}$(CR)
6. Compare(R ≈ R'), Compare(N = N')

Every time device is connected to the network and needs to communicate the Authentication phase of Algorithm 1 is executed. AA randomly chooses one of the challenges C and sends it together with the nonce value N to the device to be authenticated. The nonce value is used to prevent simple replay attacks and allows each challenge-response pair to be used repeatedly. On the device that is being authenticated the appropriate PUF response is generated, concatenated with nonce value and encrypted with public key of the Authenticating Authority. Authenticating Authority then compares (strictly) if the decrypted nonce value N' = N. Since the PUF response may slightly vary across various measurements, a predetermined number of faulty bits in R' is tolerated. If both

matches, the device is successfully authenticated. The disadvantage of Algorithm 1 is that it only performs authentication and does not provide a cryptographic context for future communication.

Authentication of a single device (D1) to the AA without asymmetric cryptography is depicted in Algorithm 2 [1]. (The Enrollment phase is the same as in Algorithm 1, steps 1–4) This method includes generating a shared symmetric key K, which requires a stable error-free PUF output. This is achieved by using an error-correcting code (ECC), denoted in the algorithm by its functions Encode and Decode. This code must have enough redundancy and structure to correct the maximum amount of errors assumed in the PUF when operated under various conditions (voltage, temperature etc.).

Choosing a suitable ECC depends on the bit error rate and length of PUF response while meeting the required corrected output length. The computational power of the device is also a limiting factor. In the case of "lightweight" devices, simple codes (such as a repetition code) are preferable.

The helper string H is a distance from the raw PUF response R to the random codeword Encode(r). It is computed by the AA (step 4 of Algorithm 2). The device then uses it to recover the key material (step 8), and subsequently derive the key K.

Algorithm 2: Authentication of a device D1 to the AA. [1].

Authentication phase – using symmetric cipher

1. **D1** → **AA**: Call(D1)
2. **AA**: r = TRNG()
3. Choose (C, R) from DB_{D1}
4. H = R ⊕ Encode(r)
5. K = KDF(r)
6. **AA** → **D1**: Challenge C, Helper string H
7. **D1**: R' = PUF(C)
8. r = Decode(R' ⊕ H)
9. K = KDF(r)
10. **D1** ↔ **AA**: Authentication + Encryption with K

The shared key K can be used for authentication and encrypted communication, as opposed to Algorithm 1, which covers only authentication, limiting its usefulness. On the other hand, Algorithm 1 does not require the generation of a helper string, nor does it need any error correction codes.

3.2 Mutual Device Authentication

Not only the device needs to be authenticated to central authority when connected to the network, the devices must be mutually authenticated before they start to communicate, as well. Similarly, as in the previous case, central authenticating authority stores the pre-generated challenge-response pair(s), and acts as trusted 3rd party. This time though, a shared symmetric key is established between the two devices, and a conventional symmetric authenticated and encrypted session can follow afterwards. The goal is to use the PUFs in both devices D1 and D2, but not transmit any PUF response over the network. Due to the one-wayness of the hash functions used, no device gets to know other device's PUF response, even if it monitors all communication. An error correcting code is used to ensure stable PUF outputs. The codewords are selected randomly from the code space by the AA. The overall process is described in Algorithm 3 [1].

Algorithm 3: Mutual authentication of D1 and D2 using AA. [1].

Authentication phase – using symmetric cipher

1. **D1** $\rightarrow$ **AA**: Call(D1, D2)
2. **AA**: r_{D1} = TRNG()
3. r_{D2} = TRNG()
4. Choose (C_{D1}, R_{D1}) from DB_{D1}
5. Choose (C_{D2}, R_{D2}) from DB_{D2}
6. $H_{D1} = R_{D1} \oplus \text{Encode}(r_{D1})$
7. $H_{D2} = R_{D2} \oplus \text{Encode}(r_{D2})$
8. $r = \text{Hash}(r_{D1}) \oplus \text{Hash}(r_{D2})$
9. **AA** $\rightarrow$ **D1**: (C_{D1}, H_{D1}, r)
10. **AA** $\rightarrow$ **D2**: Call(D1, D2) , (C_{D2}, H_{D2}, r)
11. **D1**: $R'_{D1} = \text{PUF}(C_{D1})$
12. $r_{D1} = \text{Decode}(R'_{D1} \oplus H_{D1})$
13. $\text{Hash}(r_{D2}) = \text{Hash}(r_{D1}) \oplus r$
14. $K = \text{KDF}(\text{Hash}(r_{D1}) \,||\, \text{Hash}(r_{D2}))$
15. **D2**: $R'_{D2} = \text{PUF}(C_{D2})$
16. $r_{D2} = \text{Decode}(R'_{D2} \oplus H_{D2})$
17. $\text{Hash}(r_{D1}) = \text{Hash}(r_{D2}) \oplus r$
18. $K = \text{KDF}(\text{Hash}(r_{D1}) \,||\, \text{Hash}(r_{D2}))$
19. **D1** $\leftrightarrow$ **D2**: Authentication + Encryption with K

Let us assume that D1 wants to authenticate with D2 and set up a secure communication channel. D1 initiates the process by calling the AA with the identification of D1 and D2 (Call(D1, D2)). AA contains the complete table of challenges and responses (C_{D1}, R_{D1} etc.). An error correcting code is chosen that can correct enough errors to make the PUF response stable, with the corresponding functions Encode and Decode. AA generates two random components r_{D1}, r_{D2} from the set of preimages, and encodes

them, thereby forming randomly chosen codewords. The code length should correspond to the PUF response length. Helper strings H_{D1} and H_{D2} are created by XORing the expected PUF response (R_{D1}, R_{D2}) to the corresponding codeword. The two random components are hashed and the hashes XORed to form r.

To each of the devices, a triplet (C_{Di}, H_{Di}, r) with the challenge, helper string, and r is sent. Also, in step 10, AA relays the request for communication from D1 to D2. Each of the devices challenges its own PUF to get the response (R'_{D1}, R'_{D2}). By XORing the response with the corresponding helper string (H_{D1}, H_{D2}), resulting with a codeword with errors, which is then corrected by the Decode function. This way, each device recovers its component (r_{D1}, r_{D2}). D1 recovers the value Hash(r_{D2}) by XORing r with the hash of its r_{D1}, and vice versa. Moreover, both devices know the hashes of r_{D1} and r_{D2}, and can derive the shared key K by applying a key derivation function KDF on the concatenation of the hashes.

The hashing of r_{D1}, r_{D2} is done to hide the PUF responses from the other device. If D1 monitors the communication, it will know (C_{D1}, C_{D2}, H_{D1}, H_{D2}, r). It can recover r_{D1}, and if the hashing were not done, and r would be equal to $r_{D1} \oplus r_{D2}$ directly, D1 would compute r_{D2}, and using the helper string H_{D2}, it could discover the PUF response R_{D2}. We would have to either trust all devices in the network or use all challenges only once and discard them. In our case, because we do use hashing of r_{D1}, r_{D2}, D1 only gets Hash(r_{D2}), and the one-wayness of the hash function prevents it from discovering R_{D2}. Thus, we can reuse the challenges for future authentications.

PUF response correction code choice depends on the number of bitflips inherent in the PUF operation. The code length and codeword distance determine the number of information bits, thus the length of r_{D1}, r_{D2}, and limit the entropy contained in r. By using the same challenge with multiple random r_{Di}, we can extract more bits of entropy from the PUF. The entropy of the resulting shared key K is determined by the properties of used hash functions and KDF, and the inputs. If chosen correctly, it is as high as the entropies of r_{D1}, r_{D2}. The key K is always derived from randomly chosen codewords, and therefore for the same PUF challenges (C_{D1}, C_{D2}), a different K is obtained.

3.3 Secure Communication Using Asymmetric Encryption Scheme

An asymmetric scheme provides us with benefit that there are no shared keys and the keys do not need to be transferred over secure channel. Moreover, using PUF we do not need to store the secret key on the device, instead, the key is generated during initialization phase (e.g. on boot of the device or after longer period of inactivity), when needed. We propose to use the asymmetric ElGamal encryption scheme, since there are no specific requirements on private keys (such as requirement of private numbers in RSA), apart from the requirement that the private key is from an appropriate range. For digital signatures, either ElGamal or DSA, or even ECDSA can be used. In subsequent text, we assume appropriate keys are generated depending on chosen algorithms. A private-public key

pair of each device must be generated and the public key stored in the Authentication Authority. In contrary to Algorithms 1–3, no challenge-response pairs are necessary to store in the authority. The private key is not stored but deleted immediately after using for generating the public key. This process is described by Algorithm 4.

Algorithm 4: Enrollment of asymmetric key of device D1 with the AA.

Enrollment phase (*Secure environment*)

1. **AA** → **D1**: Public parameters PP, Public key PK_{AA}
2. **D1**: Store PP, PK_{AA}
3. Choose challenge C_{D1}
4. $R_{D1} = PUF(C_{D1})$
5. r_{D1} = TRNG() so that Encode(r_{D1}) is a random codeword
6. $H_{D1} = Encode(r_{D1}) \oplus R_{D1}$
7. Store (C_{D1}, H_{D1})
8. Private key $SK_{D1} = KDF(r_{D1}, PP)$
9. Public key $PK_{D1} = GENPK(SK_{D1}, PP)$
10. **D1** → **AA**: Public key PK_{D1}
11. **AA**: Store (D1, PK_{D1})

After the keys are enrolled in the Authentication Authority, any device D1 can use it to establish a secure channel by requesting a fresh and authentic public key of its peer D2 from the AA. The public key query Q1 is created by encrypting the identity of D2 and a random nonce N1 with the AA's public key. The answer A1 contains a signed triplet with the identification of D2, its public key PK_{D2}, and the nonce N1. The device D1 verifies this signature using the AA's public key PK_{AA}.

The device's own private key SK_{D1} is generated using its PUF and the previously stored challenge C_{D1} and helper string H_{D1} (steps 8 and 9 of Algorithm 5). This private key is generated only for the subsequent signing operation and then deleted. It is not stored in the device. A new symmetric key K is generated randomly using the device's TRNG and sent encrypted with the peer's public key PK_{D2} and signed with SK_{D1}. Device D2 then verifies and decrypts this message to get the symmetric key K, as described in Algorithm 5.

Algorithm 5: Mutual authentication of D1 and D2 using AA.

Authentication phase – using asymmetric cipher

1. **D1**: $N1 = TRNG()$ *Generate a random challenge to ensure freshness*
2. $Q1 = E_{PK_{AA}}(D2, N1)$
3. **D1** → **AA**: $Call(D2, Q1)$
4. **AA**: $A1 = S_{SK_{AA}}(D2, PK_{D2}, N1)$
5. **AA** → **D1**: A1
6. **D1**: $(D2', PK_{D2}, N1') = V_{PK_{AA}}(A1)$
7. $Compare(N1' = N1), Compare(D2' = D2)$
8. $R_{D1} = PUF(C_{D1})$ *Use PUF to recall D1's private key*
9. $SK_{D1} = KDF(Decode(R_{D1} \oplus H_{D1}), PP)$
10. $K = TRNG()$
11. $CK = S_{SK_{D1}}(E_{PK_{D2}}(K))$
12. **D1** → **D2**: $Call(D1, D2, CK)$
13. **D2**: $N2 = TRNG()$
14. $Q2 = E_{PK_{AA}}(D1, N2)$
15. **D2** → **AA**: $Call(D1, Q2)$
16. **AA**: $A2 = S_{SK_{AA}}(D1, PK_{D1}, N2)$
17. **AA** → **D2**: A2
18. **D2**: $(D1', PK_{D1}, N2') = V_{PK_{AA}}(A2)$
19. $Compare(N2' = N2), Compare(D1' = D1)$
20. $R_{D2} = PUF(C_{D2})$ *Use PUF to recall D2's private key*
21. $SK_{D2} = KDF(Decode(R_{D2} \oplus H_{D2}), PP)$
22. $K = D_{SK_{D2}}(V_{PK_{D1}}(CK))$
23. **D1** ↔ **D2**: Authentication + Encryption with K

3.4 Secure Communication

After the authentication process described in the previous section, a shared key is established. At this point, a conventional symmetric authentication and session key derivation process can be performed using block ciphers such as AES. Several lightweight block ciphers suitable for embedded systems or sensor networks has been proposed, such as PRESENT [3, 28] with an 80-bit key. This allows generating the key in a single run of PUF circuit for most of the PUF designs and implementations, with no further stretching needed.

All presented algorithms in this Section utilized only PUF on the side of the devices and TRNG was used on AA. TRNG functionality on the devices is used after the secure

channel establishment (steps 10 and 19) in dependence on the communication protocols. Random numbers are needed in many classical authentication protocols [33] as well as modern internet standards such as DTLS [46].

4 Feasibility Review and Testing

As results from the previous paragraphs, a combined PUF/TRNG circuit seems to be a suitable alternative for the purpose of key generation and authentication in lightweight cryptographic applications, such as IoT devices and other embedded platforms. Such PUF/TRNG based on Ring Oscillators – ROPUF circuit was presented in our previous work [6, 7, 25, 26], so the idea of the single RO circuit can be used both for PUF and TRNG generation was validated. This circuit is depicted in Fig. 3. As the design of the protocols we present relies on the module that allows secure and efficient generation of both TRNG and PUF. We used the module to test the implementation feasibility of proposed protocols.

In order to validate the proposed authentication process outlined in Sect. 3, we performed an experiment on one device containing the ROPUF design [6, 7, 25, 26]. For this purpose, we used a ROPUF design that consisted of 2 groups of ring oscillators (ROs), each group contained 150 ROs. Only ROs from different groups were selected to form a pair, which was then used to generate part of the PUF response. We extracted 3 bits from each RO pair and enhanced the stability of the PUF output by applying Gray code on these bits [26]. The selected bits from all of the RO pairs are concatenated to create the PUF response.

In the first case, we generated the PUF responses from 150 pairs of ROs (each RO from each group was used only once), in the other, each RO was used five times (one RO from the first group is paired with 5 ROs from the other group) resulting in 750 RO pairs. These two setups achieved 450 and 2250 bits of PUF response respectively. In both cases, we performed 1000 measurements, from which we obtained a majority PUF response - R_{Di} (we determined the majority for each position of the PUF output).

In our experiment, the block length of 9 bits proved to be sufficient for the repetition code. In order to create the helper string $H_{Di,}$ we need to generate 50 or 250 random bits (r_{Di}) that are then encoded by the repetition code and XORed with the major PUF output, forming the helper string H_{Di}. This process is related to steps 2 and 4 in Algorithm 2.

The example using a simple repetition code with 5-bit block length is depicted in Fig. 4. On the device, the PUF generates a response R'_{Di} that is corrected by the helper string H_{Di}, corresponding to steps 7 and 8 in Algorithm 2. After correction, we obtained 50 and 250 bits respectively. These bits can be used to create a cryptographic key. For Algorithm 2, we can simply represent KDF as the selection of the first 128 bits (from r_{Di}) for symmetric cipher AES.

The same can be applied for Algorithm 3, where two devices are authenticating each other. However, this algorithm is more complex, since it requires implementation of suitable hash function. In case of Algorithm 1, no KDF is needed, since the AA's public key is stored on the device and PUF is not used to derive any cryptographic key.

Implementation of more efficient error would allow to increase the number of bits in generated bitstream after correction even more.

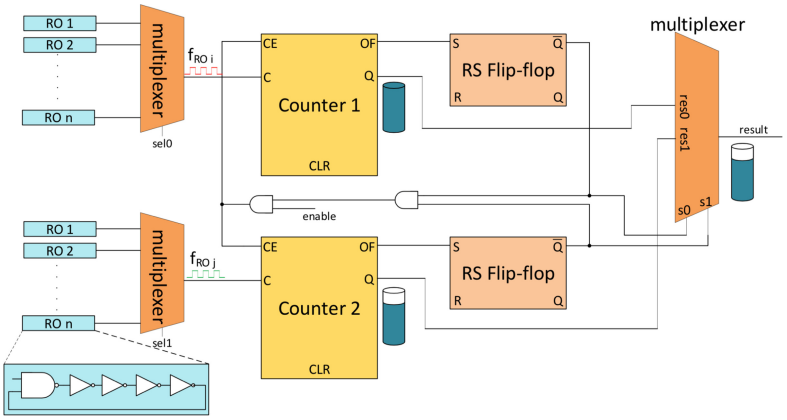

Fig. 3. PUF/TRNG circuit based on Ring Oscillators, serving as basic building block for proposed authentication and secure communication scheme [7, 25].

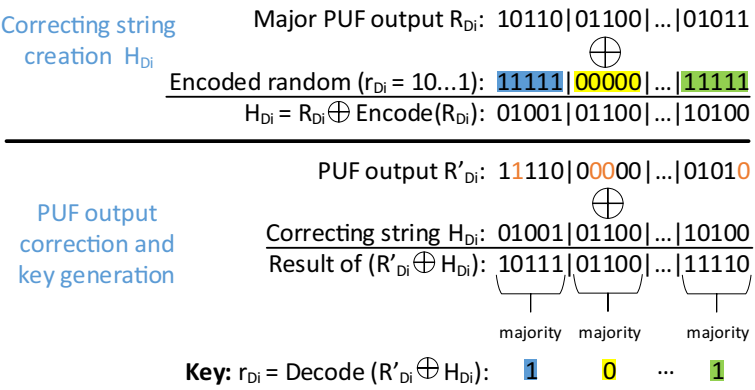

Fig. 4. Example of a simple repetition code with 5-bit groups [1].

In Fig. 5, a case study of the enrollment phase for the asymmetric key derivation according to Algorithm 4 is presented. In this case, we consider 750 ring oscillator pairs, using 3 bits from each pair [26]. This yields 2250 bits of raw PUF response, of which we use 2200 bits for key generation, using a 11 times repetition code. For this purpose, a 2200-bit helper string was derived from a 2200-bit random codeword. For this example, we consider a 200-bit private key SK of ElGamal asymmetric scheme (which corresponds to approx. 100-bit equivalent symmetric key size). The challenge C and helper string H are subsequently used to regenerate the same SK in Algorithm 5, lines 8, 9, 20, 21. The attained key lengths and the fact that the asymmetric scheme is used only once for the authentication correspond with the intended use for authentication and secure communication of IoT devices.

The experiment showed and confirmed that it is possible to generate key material for the proposed protocols, using the state of the art PUF/TRNG designs, in sufficient length and quality.

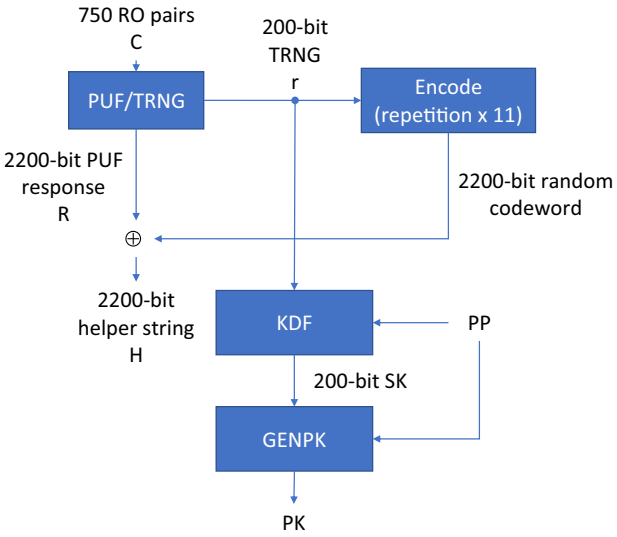

Fig. 5. Example of asymmetric key derivation for enrollment.

5 Conclusions

For the security of embedded systems, IoT and constrained devices, it is inevitable to implement the algorithms that will enable the secure authentication before the device is allowed to access the network; and secure communication to protect the confidentiality of transmitted data afterwards. In this paper we presented a set of lightweight algorithms that enable these goals even with constrained hardware, while avoiding the need to store secrets on the devices.

Several variants of authentication and secure communication protocols are presented in the paper – authentication against central authority, mutual device authentication using symmetric scheme and asymmetric encryption scheme suitable for key exchange. The combination of the protocols allows establishment of a secure communication channel – first, the devices are authenticated, then the shared session key is established, and finally, the communication may take place securely. The proposed protocols are built on PUF and TRNG as basic cryptographic primitives, as they can be efficiently implemented even in constrained devices.

Finally, a practical experiment was performed. By implementing the proposed design, following up with measurement of generated PUF responses and TRNG bistream, the feasibility of the proposed ideas was validated and confirmed. Our feasibility study and testing is utilizing the implementation a circuit combining TRNG and PUF into a unified module based on ring oscillators as it was presented in our previous work [6, 7, 25, 26] that was proved to be implementation and resource consumption efficient.

The implementation of the security measures in simple IoT devices or embedded devices is often neglected, as it is generally perceived as implementationally complex and resource exhaustive. In this paper we presented the approach and the methods that

may be used to enhance the security posture of such simple devices with constrained hardware resources, without significant overhead.

Acknowledgements. The authors acknowledge the support of the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/ 0000765 “Research Center for Informatics”.

References

1. Buchovecká, S., Lórencz, R., Buček, J., Kodýtek, F.: Lightweight authentication and secure communication suitable for IoT devices. In: Proceedings of the 6th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP, pp. 75–83. ISBN 978-989-758-399-5 (2020). https://doi.org/10.5220/0008959600750083
2. Aysu, A., Gulcan, E., Moriyama, D., Schaumont, P., Yung, M.: End-to-end design of a PUF-based privacy preserving authentication protocol. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 556–576. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_28
3. Bogdanov, A., et al.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_31
4. Bolotnyy, L., Robins, G.: Physically unclonable function-based security and privacy in RFID systems. In: Fifth Annual IEEE International Conference on Pervasive Computing and Communications. PerCom 2007. IEEE (2007)
5. Bucci, M., Germani, L., Luzzi, R., Trifiletti, A., Varanonuovo, M.: A high-speed oscillator-based truly random number source for cryptographic applications on a smart card IC. IEEE Trans. Comput. **52**(4), 403–409 (2003)
6. Buchovecká, S., Kodýtek, F., Lórencz, R., Buček, J.: True random number generator based on ROPUF circuit. In: 2016 Euromicro Conference on Digital System Design (DSD). IEEE (2016)
7. Buchovecká, S., Kodýtek, F., Lórencz, R., Buček, J.: True random number generator based on ring oscillator PUF circuit. Microprocess. Microsyst. **53**(2017), 33–41 (2017)
8. Chan, H., Gligor, V.D., Perrig, A., Muralidharan, G.: On the distribution and revocation of cryptographic keys in sensor networks. IEEE Trans. Dependable Secure Comput. **2**(3), 233–247 (2005)
9. Deak N., Gyorfi T., Marton K., Vacariu L., Cret, O.: Highly efficient true random number generator in FPGA devices using phase-locked loops. In: 20th International Conference on Control Systems and Computer Science, pp. 453–458. IEEE (2015)
10. Delvaux, J., Gu, D., Schellekens, D., Verbauwhede, I.: Secure lightweight entity authentication with strong PUFs: mission impossible? In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 451–475. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_25
11. ElGamal, T.: A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theor. **IT-31**(4), 469–472 (1985)
12. Epstein, M., Hars, L., Krasinski, R., Rosner, M., Zheng, H.: Design and implementation of a true random number generator based on digital circuit artifacts. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 152–165. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45238-6_13
13. Fairfield, R.C., Mortenson, R.L., Coulthart, K.B.: An LSI random number generator (RNG). In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 203–230. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_18

14. Fischer, V.: A closer look at security in random number generators design. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 167–182. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29912-4_13
15. Fischer, V., Drutarovský, M.: True random number generator embedded in reconfigurable hardware. In: Kaliski, B.S., Koç, çK., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 415–430. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_30
16. Golic, J.D.J.: New methods for digital generation and postprocessing of random data. IEEE Trans. Comput. **55**(10), 1217–1229 (2006)
17. Gyorfi, T., Cret, O., Suciu, A.: High performance true random number generator based on FPGA block rams. In: International Symposium on Parallel and Distributed Processing. IPDPS 2009, pp. 1–8. IEEE (2009)
18. Hammouri, G., Öztürk, E., Sunar, B.: A tamper-proof and lightweight authentication scheme. J. Pervasive Mob. Comput. **6**(4), 807–818 (2008)
19. Handschuh, H., Schrijen, G.J., Tuyls, P.: Hardware intrinsic security from physically unclonable functions. In: Sadeghi, A.R., Naccache, D. (eds.) Towards Hardware-Intrinsic Security. ISC. Springer, Heidelberg. https://doi.org/10.1007/978-3-642-14452-3_2
20. Haroon, A., Akram, S., Shah, M.A., Wahid, A.: E-lithe: a lightweight secure DTLS for IoT. In: 2017 IEEE 86th Vehicular Technology Conference (VTC-Fall), pp. 1–5. IEEE (2017)
21. Katzenbeisser, S., Kocabaş, Ü., Van Der Leest, V., Sadeghi, A.R., Schrijen, G.J., Wachsmann, C.: Recyclable PUFs: logically reconfigurable PUFs. J. Cryptogr. Eng. **1**(3), 177–186 (2011)
22. Kerckhoffs, A.: La cryptographie militaire. J. des sciences militaires **9**, 538 (1883)
23. Kirkpatrick, M.S., Bertino, E., Kerr, S.: PUF ROKs: generating read-once keys from physically unclonable functions. In: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research. ACM (2010)
24. Kocabaş, Ü., Peter, A., Katzenbeisser, S., Sadeghi, A.-R.: Converse PUF-Based authentication. In: Katzenbeisser, S., Weippl, E., Camp, L.J., Volkamer, M., Reiter, M., Zhang, X. (eds.) Trust 2012. LNCS, vol. 7344, pp. 142–158. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30921-2_9
25. Kodýtek, F., Lórencz, R.: A design of ring oscillator based PUF on FPGA. In: 2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS). IEEE (2015)
26. Kodýtek, F., Lórencz, R., Buček, J.: Improved ring oscillator PUF on FPGA and its properties. Microprocess. Microsyst. **47**, 55–63 (2016)
27. Kohlbrenner, P., Gaj, K.: An embedded true random number generator for FPGAs. In: Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays. ACM (2004)
28. McKay, K.A.: Report on Lightweight Cryptography – NIST publication (2017). https://doi.org/10.6028/NIST.IR.8114
29. Maes, R.: Physically Unclonable Functions. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-642-41395-7
30. Maes, R., Van Herrewege, A., Verbauwhede, I.: PUFKY: a fully functional PUF-Based cryptographic key generator. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 302–319. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_18
31. Majzoobi, M., Rostami, M., Koushanfar, F., Wallach, D.S., Devadas, S.: Slender PUF protocol: a lightweight, robust, and secure authentication by substring matching. In: IEEE Symposium on Security and Privacy (SP), pp. 33–44 (2012)
32. Malina, L., Hajny, J., Fujdiak, R., Hosek, J.: On perspective of security and privacy-preserving solutions in the Internet of Things. Comput. Netw. **102**, 83–95 (2016)
33. Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)

34. Öztürk, E., Hammouri, G., Sunar, B.: Towards robust low-cost authentication for pervasive devices. In: IEEE Conference on Pervasive Computing and Communications, PerCom (2008)
35. Pappu, R., Recht, B., Taylor, J., Gershenfeld, N.: Physical one-way functions. Science **297**(5589), 2026–2030 (2002)
36. Raza, S., Shafagh, H., Hewage, K., Hummen, R., Voigt, T.: Lithe: Lightweight secure CoAP for the Internet of Things. IEEE Sens. J. **13**(10), 3711–3720 (2013)
37. Roman, R., Zhou, J., Lopez, J.: On the features and challenges of security and privacy in distributed internet of things. Comput. Netw. **57**(10), 2266–2279 (2013)
38. RSA Laboratories: PKCS #5 V2.1: Password Based Cryptography Standard (2012)
39. Schindler, W.: Random number generators for cryptographic applications. In: Koç, Ç.K. (ed.) Cryptographic Engineering. Springer, Boston (2009). https://doi.org/10.1007/978-0-387-71817-0_2
40. Schleiffer, C., Wolf, M., Weimerskirch, A., Wolleschensky, L.: Secure key management-a key feature for modern vehicle electronics. Technical Report, SAE Technical Paper (2013)
41. Sicari, S., Rizzardi, A., Grieco, L.A., Coen-Porisini, A.: Security, privacy and trust in Internet of Things: the road ahead. Comput. Netw. **76**, 146–164 (2015)
42. Sklavos, N., Zaharakis, I.D.: Cryptography and security in Internet of Things (IoTs): models, schemes, and implementations. In: IEEE Proceedings of the 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS 2016), Larnaca, Cyprus (2016)
43. Suh, E.G., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Proceedings of the 44th annual Design Automation Conference, pp. 9–14. ACM (2007)
44. Suh, E.G., O'Donnell, C., Devadas, S.: AEGIS: a single-chip secure processor. IEEE Des. Test Comput. **24**, 6 (2007)
45. Tkacik, T.E.: A hardware random number generator. In: Kaliski, B.S., Koç, çK., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 450–453. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_32
46. Tschofenig, H., Fossati, T.: Transport Layer Security (TLS)/Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things. RFC 7925, July 2016
47. Van Herrewege, A., et al.: Reverse fuzzy extractors: enabling lightweight mutual authentication for PUF-Enabled RFIDs. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 374–389. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_27