μSystems Research Group
School of Engineering

PUF-Based Authority Device Scheme

Konstantinos Goutsos

Technical Report Series
NCL-EEE-MICRO-TR-2019-212

July 2019 (Revised)

Contact: k.goutsos1@ncl.ac.uk

NCL-EEE-MICRO-TR-2019-212

Published: May 2019
Revised: July 2019

μSystems Research Group
School of Engineering
Merz Court
Newcastle University
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

PUF-Based Authority Device Scheme

Konstantinos Goutsos

Abstract

With the rise of the Internet-of-Things, followed a tendency to create unified architectures with a great number of edge nodes and inherent security risks due to centralisation. At the same time, security and privacy defenders advocate for decentralised solutions which divide the control and the responsibility among the entirety of the network nodes. However, spreading the responsibility among a great number of parties also leads to increased risk of leakage for secret information.

A solution to achieving the best of both worlds could be the primitive of unclonability which forms the basis of any relationship, be it human or between devices, as it provides proof of uniqueness for the communicating entities. This uniqueness also has a direct effect on the value of an unclonable object since no other copies exist to share this value. From the IoT perspective, unclonability can offer strong security guarantees, distinction among otherwise identical edge nodes, and higher levels of control over the system by its owners.

Unclonability has been realised on a physical level via the use of Physical Unclonable Functions (PUFs) but methods to expand it to fully formed security frameworks have not been developed. In this report we attempt to set the foundations for the development of an *unclonability stack*, propagating the primitive from the unclonable chips of PUFs, to devices, network links and eventually through to unclonable systems. To that end, we also present an *Authority Device Scheme (ADS)* and discuss its security properties, along with a basic prototype.

The role of the 'authority devices' is that of a consolidated, observable root of ownership, which can be verifiably handed over or destroyed, all the while without requiring a central authority for the normal operation of the system. As such, these devices are used to bootstrap the operation of a network system and introduce network nodes to each other, enabling them to form groups or *neighbourhoods*. This is achieved via asymmetric cryptography with secrets that are generated on demand by PUFs and never saved in persistent storage. After their introduction, nodes are able to identify and interact with their peers, exchange keys and form relationships that enable novel features in the higher layers of the stack.

1 Introduction

In this report we present *Authority Device Scheme(ADS)*, a collection of cryptographic protocols based on *Physical Unclonable Functions(PUFs)* and one or more *authority devices (ADs)*. The scheme includes protocols for the introduction, mutual authentication and clustering for network nodes along with advanced features by combining the novel properties of Physical Unclonable Functions with existing key management methods and public key cryptography.

The ADS enables grouping nodes into clusters or *neighbourhoods*, making them aware of their neighbours and using this awareness as a security enabler. This is achieved through the distribution of identifiers among the nodes, through the authority devices. These devices act as a proxy both among pairs of nodes and between the system and its owner.

Researchers have proposed numerous solutions for securely introducing devices to each other and forming clusters, including the assumption of a secure environment[1] or the use of user input like PIN codes[2]. However, an entity which owns a system should be able to ascertain this *authority*, since it is only via this ownership that the system exists in its current form. In other words, the owning entity (an individual or an organisation) should have complete authority over the hardware but also the information that powers in the system.

Nevertheless, it is neither secure, nor convenient for human operators to have access to the device secrets such as cryptographic keys, identifiers etc. In our scheme, by representing the authority of the system owner with *authority devices*, the owner retains her authority over the system without being exposed to implementation secrets and specifics. As a result, the user is relieved of the burden of key provisioning and management, reducing the attack surface, and advanced features such as delegation of authority, and behaviour attestation are enabled while higher security is obtained via exploiting the *unclonability* property of PUFs. Additionally, strong guarantees can be achieved regarding the decommissioning of devices, since it is possible to revoke these authority devices. In contrast it is not feasible to revoke information, such as cryptographic keys, that has already been exposed.

In practice, unclonability can be manifested with Physical Unclonable Functions (PUFs), self-contained physical blocks producing unique outputs based on their physical features. Section 2 offers a more detailed view of unclonability, PUFs, and our vision for unclonable systems.

The proposed scheme has a dual purpose: to act as an enabler for introducing the unclonability primitive in higher level protocols but also further research around the primitive. We strongly believe that there is high value in new methods of managing the security of network systems via exploiting secret information that inherently occurs in electronic devices, without manual generation or exposure to the device's environment.

Due to the nature of the scheme, there is no need for a permanent managing authority that would create a single point of failure. In fact, if the application scenario requires it, authority devices can be physically destroyed after the initial system setup. System operation would then continue normally, albeit without the ability to make topology or identity modifications.

We also introduce a reference architecture for a cryptographic core (cryptocore) that includes a PUF. While the PUF provides an *unclonable root of trust* to the system, the core serves as a physical container, providing a secure interface to the PUF, and accelerating the required cryptographic operations.

In order to focus on a high-level view of the proposed methods, this report does not discuss specific cryptographic algorithms which would be used by the scheme. In the prototype of Section 6, a number of design decisions were made to match common practice at the time of writing. As a result, Section 6 can also be seen as guidelines that would be considered secure in the context of our work. However, it will be made clear that the ADS is fully flexible as it merely relies on the generic features of unclonability and asymmetric cryptography and can be adapted to the application or future cryptography needs.

1.1 Use Cases

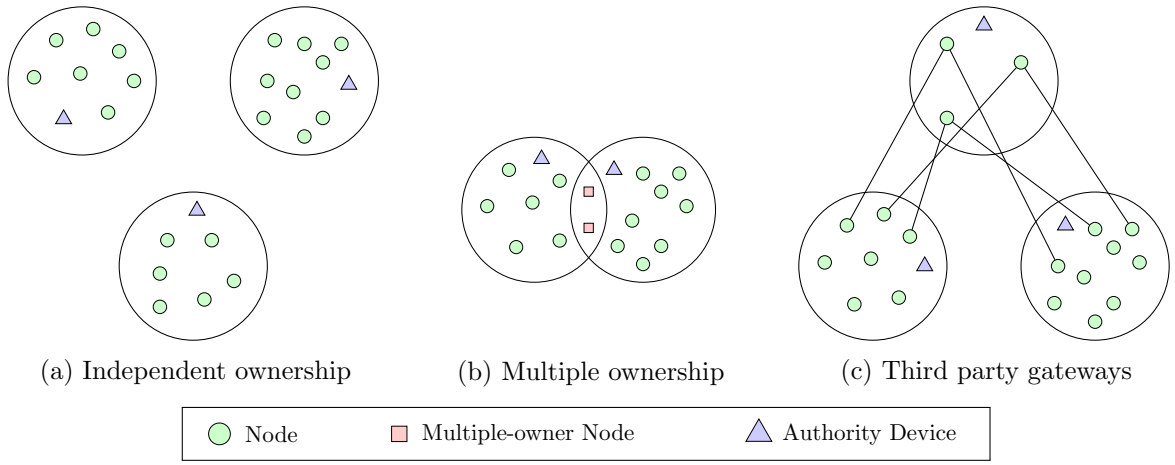
The ADS can be used in a number of topologies such as the ones pictured in Fig. 1. In the figure we can discern three main configurations, representing the majority for real world scenarios, with devices under the same authority grouped in circles.

The first case (Fig. 1a) is one where nodes under different ownership are divided into separate neighbourhoods with distinct authority devices. For example, this would correspond to a number of discrete company departments.

In the second case of Fig. 1b on the other hand, a subset of the nodes belongs to two different groups, and is under the control of two different authority devices, as is the case for the same company employees who belong in multiple teams.

Fig. 1c illustrates a hierarchy of nodes where members of one of the groups are tasked with interacting and relaying information between otherwise separate authority domains. The higher level domains interacts directly with just a small number of nodes in the lower level groups, essentially creating two layers of relays. Evidently, this case corresponds to a company with a management hierarchy and individual team leaders.

In this section we present a number of typical use cases for the proposed scheme, in order to clarify the subsequent description of our design choices.



(a) Independent ownership (b) Multiple ownership (c) Third party gateways

Figure 1: Example Topologies

1.1.1 Military Sensors

Military units often operate in a tree structure where the soldiers (tree leaves) report to higher rank officers (intermediate tree nodes) who recursively report to the next rank, continuing upwards until the commanding officer (root of the tree). Also, soldiers need to be able to perform simple tasks as instructed with the least training possible.

We can envision a situation where a number of sensors need to be deployed over a battlefield to gather and relay sensitive information. The commanding officer, having the authority over the sensors, prefers to avoid exposure and, as a result, needs to delegate the task of configuring the system to soldiers who are not experts in networking or security. At the same time, it is necessary to ensure that in case a soldier or a sensor is captured, the damage will be contained to the smallest possible organisational unit.

Using an authority device, the commanding officer is able to perform the setup process at the military base, and pass the AD on to his soldiers who will install and configure the sensors in the field. Since the history of the AD is observable, the officer can verify the correctness of the system and be certain that any unexpected behaviour will be detected.

Extending the above to multiple units with separate commanding officers, similarly to Fig. 1a, if a soldier bearing the AD for the group is captured, then the officer will be eas-

ily able to detect the threat and isolate it, being certain that no sensor secrets were stored on the AD and no other units are affected. Additionally, if a number of sensors are compromised, the remaining ones can be readily reconfigured with a new AD device, invalidating any action or communication performed by the compromised nodes.

1.1.2 Smart Cities

In smart city applications, hundreds of smart nodes are deployed over urban areas and perform operations based on coordination. These devices need to be installed in a physically secure manner but this is not always possible, due to the inability to supervise the devices and the complexity of the task for city staff. As a result, the devices are often not secure, bearing the same secrets as their peers, and waiting to be compromised.

With the ADS, we envisage applications where the supervisor of a smart city managing team will be able to give her employees an authority device and ask them to install and configure the nodes. Upon completing the installation, the city employees will simply connect the AD to each of the nodes and the configuration would be performed automatically, without exposing any secrets to any persons involved. Furthermore, the AD can keep a list of all the nodes it came in contact with, making it possible for the supervisor to verify that all nodes were set up correctly.

A similar topology involving multiple projects with different city officials running them, can be seen in Fig. 1a.

1.1.3 Corporate Computers

It is a common occurrence in corporate environments for employees to use 'off-the-shelf' computers that are reused when they are no longer needed. Also, certain employees might work on projects that involve multiple departments. As such, it would be beneficial for the IT department to be able to easily configure employee machines in a secure way.

In this scenario, resembling Fig. 1c, the same employee holds multiple authority devices, corresponding to different teams. Using the ADS a large number of company workstations can be efficiently configured to provide granular access and communication between different departments. Due to the minimal user interaction required, this configuration can be performed by team leaders and department heads, removing the need for time-consuming requests to the IT department.

Thus, it is clear that the security and usability of the company's security policy is greatly improved. Without authority devices, the IT team would have to manually install secrets to each machine and consequently be in knowledge of the secrets. Furthermore, in larger companies, managing the complexity of machines belonging to different departments is often problematic, resulting in insecure solutions of granting access to more entities that it is required.

In summary, the Authority Device Scheme delivers the following:

Improved security: All key pairs are dynamically and automatically generated based on PUFs challenged with randomised inputs. Thus, no parties outside the cryptographic core, have access to any private key material. This leads to the minimisation of trust relationships and thus, increased security since no party can be coerced into revealing device secrets.

In addition, even if an adversary obtains an authority device, the information stored on it will only allow her to perform high level operations on the system without compromising the communication of the nodes.

Reduced complexity and improved scalability: Due to the dynamic nature of the key material, the need for user interaction is reduced to simply connecting an authority device to the nodes. Additionally, on-demand key generation removes the need for secure non-volatile storage as the keys are only present while the system is powered on.

Decentralised operation: After their initial introduction, nodes operate without the need for a central authority. Thus, neither the security nor the robustness of the system depend on a single device.

Novel neighbourhood features: The ADS encompasses the first four layers of the *unclonability stack* described in Section 2 (provider, core, device, and protocols) and, when combined with the strong security properties of unclonability, enables a range of new scenarios and features. Those include the detection of *distortions* in the system (cloned, removed, moved or misbehaving nodes) and behaviour monitoring of neighbourhoods. Work on those features will be presented in future publications.

The remainder of the report is organised as follows: Section 2 discusses the theoretical and practical building blocks underpinning the development of the ADS. A reference architecture of the cryptographic core is presented in Section 2.4 and Section 3 sets the basis for the subsequent description of the ADS. The operation of the scheme is detailed in Section 4 and its security analysis in Section 5. Finally, a prototype implementation is briefly examined in Section 6 followed by the concluding Section 7.

2 Background

2.1 Unclonability

In order to formulate the notion of unclonability, we first need to define the meaning of *clone*. In the context of physical objects, clones can appear on two levels: mathematical and physical[3]. *Mathematical clones* essentially treat the original object as a black box and try to emulate its behaviour, usually generating the same mapping of inputs to outputs. *Physical clones* on the other hand, are identical copies of the physical structure of the object in detail.

There exist different levels of success in cloning an object and, in practice, it is quite difficult to achieve a perfect cloning result. Therefore, the task of cloning and that of clone detection are evaluated by the complexity needed to produce satisfactory results. To make clone detection possible (and thus achieve unclonability) one needs to draw on one or more features which are inherent to the object and beyond any level of control that would allow their exact reproduction, given the available technology.

Unclonability refers to the difficulty in controlling all the features of an object in a meaningful way, with the aim of producing a clone that is, or appears to be, an exact copy of the original object. Thus, there are the following prerequisites for an object to be unclonable:

1. The presence of *individualising features*[3] which differentiate it from other similar or dissimilar objects.
2. The ability of an observer to measure those features in a quantifiable manner and utilise the results.
3. The persistence of those features over the lifetime of the object or the time of interest.

While individualising features can vary considerably between different types of objects, all of them share some common qualities, to be considered exploitable in practice:

Unclonability: Being hard to thoroughly copy or otherwise reproduce.

Modelling resistant: Exhibiting behaviour that is hard to represent with a model.

Small intra-distance: Generating the same response over multiple observations of the same object, up to a bounded error.

Large inter-distance: Generating very different responses over multiple observations of different objects.

Observability: Providing an practical and efficient way of observation.

Stability: Retaining the same value over time.

In conclusion, it is evident that unclonability is closely related to ownership and authority, and while it sounds attractive in theory, there remains a lot to be done for it to be used in practical applications. Importantly, a big part of human interactions in modern societies takes place remotely where there is often little physical proof that the participating parties are who they claim to be. As such, there is a need to unequivocally prove one's identity while keeping it safe in an increasingly connected world. Until humans can directly interact with the digital world, intermediate "identity devices" will need to be employed to provide these features.

2.2 Unclonability Stack

Exploiting the unclonability primitive, we can construct an *unclonability stack*, a collection of layers building on unclonability to enable novel applications.

The stack is summarised in Fig. 2 and can be divided into two domains: physical and logical. The layers of the physical domain have to be implemented in and supported by hardware for various security reasons discussed below. On the other hand, the logical domain is mainly concerned with the higher level interactions of the system and can be implemented in either software or hardware. Each of the layers is providing its features to the higher layers via an interface that aims to reduce complexity and improve security by

Despite resembling the OSI model, the unclonability stack is concerned with the security interactions between devices and systems rather than the exchange of application data. Consequently, a different set of methods and protocols has to be employed to transmit and receive application data. Traditional communication protocols would be suitable and as the stack is designed to provide additional security features to existing communication infrastructure.

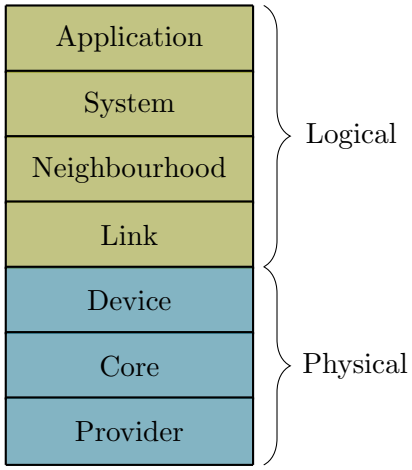


Figure 2: Unclonability Stack

Provider: A low-level hardware construction that provides physical unclonability. In our work, Physical Unclonable Functions, as they are described in Section 2.3, serve as providers.

Core: With the provider as its centrepiece, the core provides cryptographic operations such as key generation, encryption, and signing, to the higher layers. It also provides access to the unclonability Provider through a *secure interface*. The purpose of the core is to prevent exposure of the internal secrets of the Provider via physical and logical separation. The aforementioned secure interface facilitates the logical separation by defining strict rules for a small number of simple operations. By keeping the operations of this layer as well-defined as possible, the attack surface for the Core is reduced, and attack attempts can be detected.

Device: The Device layer represents a networked device (node) that includes a Core. On a conceptual level, the boundaries between the Core and the Device can be fuzzy but, in essence, the Device usually has more complex functionality to serve the purpose of the application. This functionality is unrelated to the security aspects of the Device and often expands the attack surface for potential adversaries. As a result, logical and physical separation between the Core and Device is not needed but highly advisable.

Link: The Link layer includes protocols that enable the establishment of identity between nodes, the detection of distortions etc.

Neighbourhood: Nodes are organised in clusters or 'neighbourhoods'. Due to the features provided by the Link layer, neighbourhoods can be established and distortions can be detected on a collective level, enabling the monitoring of neighbouring nodes and the response to attacks. In addition, neighbourhoods can exploit their topology to provide more security features, including among others redundant paths and traffic pattern obfuscation.

System: The *System* comprises a number of neighbourhoods which interact through strict protocols. These protocols are designed to enable inter-neighbourhood communication while retaining high levels of security by transforming and routing packets appropriately.

Application: Leveraging the inherent unclonability of the system, application developers can create novel scenarios or drastically improve the security of existing implementations.

2.2.1 Unclonable Device

Atomicity is exceptionally difficult to achieve in networked systems where devices are assumed to be identical to each other, forming a swarm of perceived clones. While this assumption simplifies the development of large systems, it also hinders the operation of sophisticated security methods. By being unable to differentiate between nodes, security protocols can only distinguish among classes of devices rather than individual ones.

In the past, identifying these devices was based on some kind of secret that had to be generated, safely stored and recalled every time it was used. However, in a lot of cases, there are no safeguards in place to prevent the secret from being copied, allowing the impersonation of the original device. Some attempts to prevent this cloning have been made in the past but most of them were eventually proven insecure or impractical due to high implementation costs.

In IoT scenarios, the mathematical guarantees provided via cryptographic means are becoming increasingly irrelevant since attackers have access to the device hardware, allowing them to recover secrets by physical attacks. Furthermore, a lot of applications involve a great number of devices, making the process of generating and storing unique secrets inefficient. An unclonable node is able to provide a secure and high-entropy method of generating a unique secret on the device itself, recreating it every time it is needed, without the need for secure storage. Additionally, the use of unclonable network nodes provides the ability to impede a further class of attacks. For example, since device identity can be unquestionably proven, a number of *topology distortions* can be detected, as discussed in Section 2.2.2.

An *unclonable device* is defined as a generic computing device with components summarised below. A reference architecture with these component is shown in Fig. 3.

Unclonability Core: A self-contained block, with the features and architecture discussed in Section 2.4, providing a *root of unclonability* to the device, as well as performing the necessary cryptographic operations for the ADS.

General Purpose Logic: Capable of performing the operations required by the application.

ADS Logic: Implementing the Authority Device Scheme.

Communication Interfaces: Networking and other I/O interfaces, employed by the ADS and the application logic.

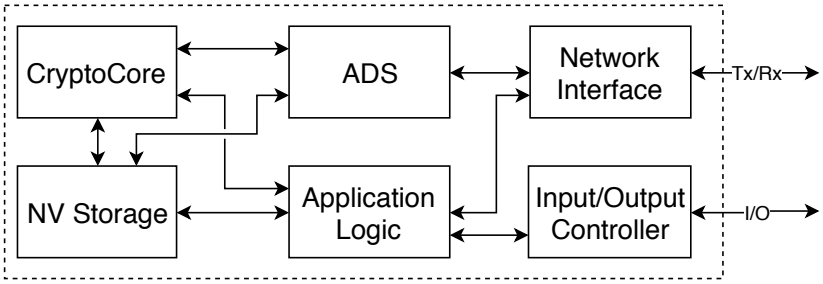


Figure 3: Unclonable Device Reference Architecture

Exploiting this architecture, an unclonable device is able to securely generate keys that are bound to the intrinsic physical properties of the hardware instance, and execute the required operations to support the features of higher level schemes including the ADS. Acting as an *unclonable root of trust* via undeniably proving its identity, the device is able to participate in, and organise unclonable network links and neighbourhoods.

2.2.2 Unclonable Neighbourhoods

The higher layers of the unclonability stack are not in the scope of this report. However, in this section we will attempt to provide a view of the concept of *unclonable neighbourhoods*, to clarify the goals of the Authority Device Scheme.

In networking terms, neighbourhoods of nodes are equivalent to clusters. As is the case in a town neighbourhood, neighbours need to be introduced to each other to form trust relationships. Nodes that are introduced to each other assume that the remote node is approved by a higher level entity (possibly a central node) that is eventually controlled by the owner of the system. In practice, a few layers of ownership relationships can exist between the owner and the nodes, i.e. system owner $\rightarrow$ system administrator $\rightarrow$ authorisation token $\rightarrow$ node.

Neighbourhood unclonability can be realised with protocols which take into account both the topology of the system and the relationships among neighbouring nodes. In essence, by extending the unclonability properties of individual nodes, we can envisage unclonable links between nodes leading to unclonable clusters of devices.

Based on unclonable neighbourhoods we can create a system model where any *topology distortions* can be detected. Distortions modify the structure of the network and can be the result of an attack by an adversary or simply a modification by a legitimate source. We can discern a number of basic cases of distortions:

Node removed An enrolled (known) node is removed from the network.

Node replaced A known node is replaced by an unknown, possibly malicious or compromised node.

Node introduced A unknown node is introduced, being either honest or malicious.

Node moved A known node is moved to a different logical position.

Neighbourhood unclonability protocols should detect the above distortions and take actions that might include alerting a higher level party, destroying secrets, initiating recovery procedures etc. Nonetheless, it is important to make the distinction between distortion detection and distortion recovery. For the purposes of security, it is acceptable and even *desirable* for the system to seize operating normally after a distortion occurs, requiring an exceptional 'authority action' for recovery. However counter-intuitive, this feature ensures that the system will operate in a predictable manner even in case of an attack.

2.3 Physical Unclonable Functions

The most prominent practical embodiment of unclonability are *Physical Unclonable Functions (PUFs)*. Initially proposed in an analogue form by Pappu[4], PUFs make use of intrinsic variations in the fabrication process of hardware components, to accept a challenge and produce a response. In other words they are functions with two inputs: the challenge and the physical features of the physical block. Evidently, PUFs are an efficient way to harness a secret from the physical domain and use it repeatedly without being able to alter or copy it.

While many types of PUFs exist, electronic PUFs have been the focus of most research efforts, due to their efficiency and low cost. Thus in our work, we only consider electronic PUFs which have digital inputs and outputs making it possible to integrate them in algorithms and methods designed for binary operands. At the same time, electronic PUFs can be constructed out of components that are often encountered in digital systems like SRAM chips, arbiters, XOR gates, delay circuits etc.[5]–[7]

In the context of our work, PUFs are able to provide the following advantages which, in many cases, intertwine:

Core of unclonability Evidently, PUFs can serve as the main primitive in scenarios involving unclonability.

High quality cryptographic keys Due to their high entropy output and their inherent physical protection, the responses of PUFs can be used in the place of regular cryptographic keys. It is also possible to regenerate the keys on demand without direct user interaction, making secure storage unnecessary.

Trust relationship minimisation Users and administrators of PUF-enabled hardware do not have access to the cryptographic secrets of the system. Thus, due to the removal of the human element, trust relationships can be minimised, leading to a reduced attack surface and a higher cost of attack.

2.3.1 Definition

PUFs can be formalised as a mapping $\tau : C \rightarrow R : \tau(c) = r, c \in C, r \in R$ generating the challenge-response pairs(CRPs) $(c_i, r_i) = (c_i, \tau(c_i))$ and exhibiting the following properties[3], [8]:

1. *Evaluable*: given τ and challenge c, it is fairly easy to get the response $r = \tau(c)$.
2. *Unclonable*: given τ it is hard to construct a procedure $\gamma : \forall c \in C, \gamma(c) \approx \tau(c)$ up to a small error.
3. *Unpredictable*: Given a set Q of CRPs: $Q = \{c_i, r_i\}$ it is hard to compute $r_u = t(c_u), c_u \notin Q$
4. *Unique*: $\forall c \in C, \tau(c)$ is uniquely mapped to the physical entity in a way that no other entity, however similar will present the same CRP even up to a small error.
5. *One way*: Given only r and τ it is hard to find $cst.r = \tau(c)$
6. *Tamper evident*: Any intrusion attempt to the physical construction is highly likely to transform τ to $\tau' st. \forall c \in C : \tau(c) \neq \tau'(c)$ with this difference being significant and detectable.
7. *Reproducible*: For a set $c \in C, r = \tau(c)$ is reproducible with a small, bounded error.

Due to their properties, PUFs can be used as a building block in a variety of security scenarios and protocols including identification[4], [9], authentication[2], [10], [11], signature[12], key generation and storage[13]–[15], and key exchange[4].

2.3.2 Taxonomy

In the recent years a number of efficient and highly secure PUF constructions have been proposed with the most notable being SRAMs[5], Arbiters[6], and Ring Oscillators[7]. Most electronic PUF constructions can be adapted to provide the Input/Output behaviour required by our scheme, and thus reviewing specific implementations is out of the scope of our work.

An important distinction of PUF constructions is between *strong* and *weak* PUFs[16]. A strong PUF has a infinite challenge-response space and its responses are completely unpredictable and uncorrelated. The former property is especially important in authentication scenarios where a large number of unique CRPs is used. Despite the fact that no practical strong PUF construction has been proposed, various protocols in literature are based on strong PUFs. Our scheme avoids the assumption of a large CRP space and strives to reduce the number of CRPs required throughout its protocols.

A number of more sophisticated PUFs have also been proposed. *Reconfigurable PUFs* allow one-way updating of their internal state, invalidating all previous CRPs[17], [18]. *Erasable PUFs* provide both a large CRP space and are able to erase specific CRPs from their state[19]. Finally, *matched PUFs*, are based on creating two distinct ICs that will have the same behaviour (up to an acceptable error) but at the same time making it statistically impossible for a third matching IC to be created[20].

Unfortunately, physical complexity is both the major advantage and a hindrance when it comes to implementing sophisticated unclonability solutions. As a result, most proposals discussed above have limited practical value, until they are developed further.

2.3.3 PUF Authentication and Identification Protocols

By definition, authentication and identification protocols involve two parties: the prover and the verifier. Typically, the verifier aims to establish that the prover possesses some required attributes (i.e. what the prover has, knows, or is[21]) via a series of queries. These queries are designed to demonstrate the possession of the aforementioned attributes with a high probability.

Most methods of utilising PUFs in authentication protocols consists of two phases: enrolment and verification. It the simplest case, during enrolment the prover's PUF is queried with a number of random challenges and the resulting challenge-response pairs (CRPs) are stored in a database on the verifier. In the verification phase, the verifier picks a random CRP from its database and sends the challenge to the prover who is required to reproduce the response generated during the enrolment phase to be successfully authenticated. This method has a number of drawbacks including large storage requirements, exposure of the PUF responses, and placing a great amount of trust in the verifier.

A number of works based on this method have been published, including Gassend et al.[9], and Devadas et al.[22]. More recent publications [23] extend the classic CRP technique with the integration of additional physical features but they do not effectively alleviate any of the method's drawbacks.

To make matters worse, PUF responses are rarely perfectly stable, with an error rate of 7% to 20%[3], resulting in the need for computationally intensive error correction methods. In the case of simple identification protocols however, it is possible to completely avoid using error collection logic and identify the PUF devices with an appropriately calibrated difference threshold. Of course, the main drawback of this method is the difficulty in specifying a threshold that would disallow false positives in critical applications.

In order to improve the error correction performance, Van Herrewege et al.[24] relocated the expensive error correction calculations to the verifier device which is typically more powerful than the prover. Their method provides mutual authentication while the CRP pairs are never publicly revealed and thus there is no need for them to be discarded after every protocol round. Nevertheless, the need for a large CRP database is still present in this solution, with the associated scalability issues.

A different class of PUF protocols is based on the ability of the verifier to construct and securely store a model of the prover's PUF. A number of solutions have been proposed based on this idea, including Slender[25] which uses fuses to physically disable the PUF's modelling interface after enrolment, and (SIMPL) systems[26] which are based on the assumption that a model will always be orders of magnitude slower that the real PUF. However, the latter has not been implemented in practice, due to the difficulty of guaranteeing the validity that assumption.

Finally, algebraic schemes making use of more complex mathematical structures have been proposed, in order to achieve specific security goals. A representative example is the method proposed by Krzywiecki in [21], using Lagrangian interpolation to allow anonymous authentication of users equipped with a PUF device.

2.4 Cryptographic Core

The ADS is based on a set of security primitives: public key cryptography, physical unclonable functions, and physical security. For a correct and secure operation, a minimal set of requirements has to be satisfied, and we find that these requirements would be best supported by a separate cryptographic core that will enable both physically secure and efficient operation. A reference architecture of this core is presented in Fig. 4 and discussed in this section. In summary, an implementation of the core would require the following minimum features:

- Non-volatile memory for key challenges and helper data.
- PUF block with an externally accessible secure interface, to generate (a) unclonable seeds for key generation, and (b) PUF CRPs to be used in the ADS protocols.
- Cryptographic processor supporting public key cryptography, mainly key generation and signature generation/verification.
- True random number generator (TRNG) to generate the random tokens required by the scheme.

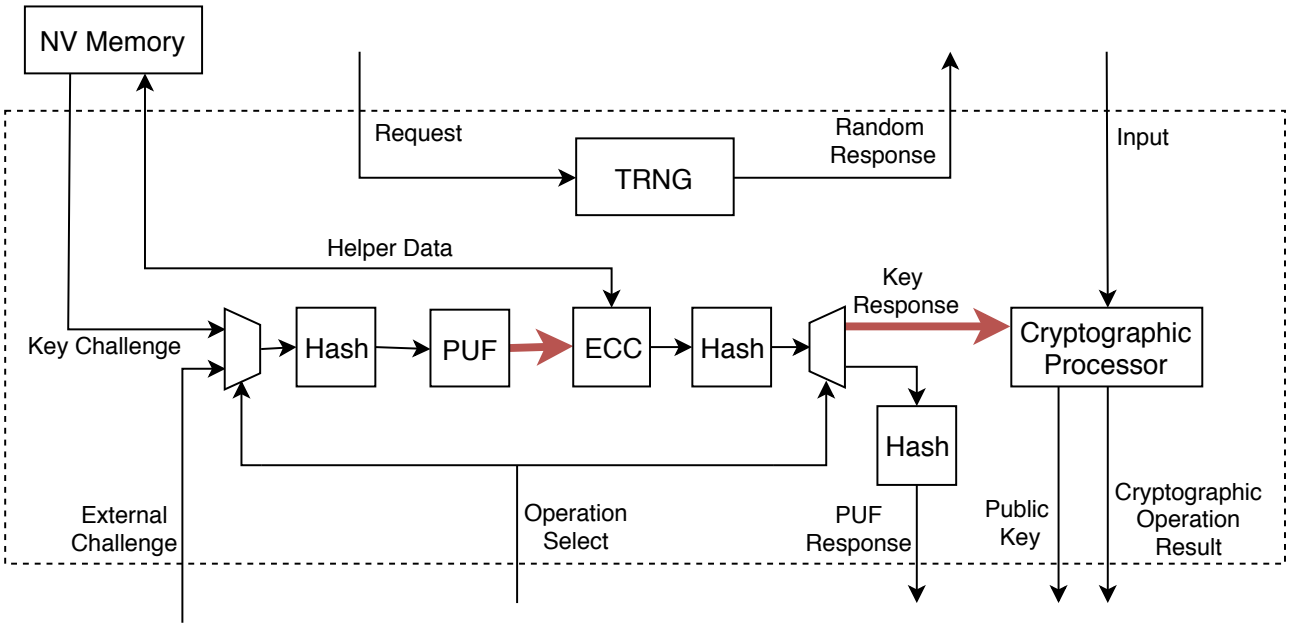


Figure 4: Cryptographic Core Reference Architecture

2.4.1 Operation Modes

The scheme requires two main functions from the cryptographic core: generating and verifying cryptographic signatures, and providing a secure interface to the PUF block. In the first mode, the private key for the cryptographic part is generated by the PUF, using a challenge stored in memory. Subsequently the key is used in the cryptoprocessor for the required tasks but it is never stored in memory as it can be regenerated on demand. In the second mode of operation, the core accepts arbitrary inputs, uses them as challenges to the PUF and returns the responses after they are sufficiently post-processed to enhance security.

2.4.2 Non-Volatile Memory

Two types of data need to be stored in non-volatile memory: key challenges and PUF helper data. The key challenges are used as inputs to the PUF to produce the private keys while the helper data are produced (and subsequently consumed) by the error correction (ECC) block to enable the error elimination of unstable bits in the PUF response. As per the adversary model of Section 3, the data written in persistent storage is considered public, thus even a memory block that is shared with the rest of the system would meet the requirements.

2.4.3 PUF

The PUF block is an abstract representation of an electronic PUF construction. For the purposes of the ADS, the PUF is treated as a black box that accepts challenges and provides the appropriate responses as discussed in Section 2.3. The rest of the cryptographic core makes use of these responses as instructed by the protocols of the ADS.

As seen in the following sections, PUF responses are used in the ADS either as a seed for key generation or directly in authentication protocols. This results in some PUF responses being made available to third parties and even adversaries. Thus, the cryptocore needs to ensure that challenges that have been used in the past for key generation are not used for any other purpose. This is achieved by simply hashing the PUF output once more, if it is to be shared outside the boundaries of the cryptocore.

2.4.4 Error Correction and Hashing

The pair of Error Correction Code (ECC) and Hash blocks ensures that the PUF response is both stable and secure, by performing error correction and obfuscating the raw PUF output. To enable the error correction coding, helper data are produced and stored when a response is generated for the first time. The data is retrieved and used as an input to the error correction algorithm when the same response needs to be reconstructed. Various error correcting methods have been proposed in literature (e.g. [3], [11], [13], [27]) with different advantages and disadvantages. An important consideration for selecting an error correction method is the possible leakage of PUF entropy through the helper data, since the latter are stored in unprotected memory.

Before being released to the output, PUF responses are hashed with a cryptographic hash function, enhancing the entropy of the responses, impeding modelling attempts against the PUF, and, as discussed above, ensuring the secrecy of the key material. In practice, both the input and output of the PUF might be hashed, compressing them to pre-defined sizes.

The details of error correction and hashing are not discussed, as we are mainly interested in higher level protocols, but have been extensively presented in literature along with PUF operation specifics[3], [11], [13], [27]. The work presented in Section 6 makes uses of state-of-the-art techniques to simulate the cryptocore in software but these techniques could be replaced by future research developments as long as the I/O behaviour of the blocks remains unchanged.

2.4.5 Cryptographic processor

The cryptographic processor (cryptoprocessor) performs three main tasks in the Authority Device Scheme: key generation, signature generation, and signature verification. Thus, the cryptoprocessor is required the support public key cryptography. By implementing all the blocks of the cryptocore on a single chip, physical security measures can be taken against side channel attacks. To allow for the use of the cryptoprocessor by other applications besides the ADS, an interface to the cryptoprocessor itself is provided, with the restriction of accepting and producing strictly public data.

2.4.6 True Random Number Generator

Throughout the scheme a number of random nonces are employed to ensure the freshness of the protocols and it is required that these nonces are not repeated over the lifetime of the system, to prevent replay attacks. Thus the cryptocore includes a True Random Number Generator (TRNG). An additional PUF component can possibly serve as a TRNG as seen in literature[28].

This requirement, at least in the context of the ADS, can be eliminated if other methods of replay attack prevention are used, such as including session counters to the exchanged nonces. However, such information would require the verification of its integrity before use, if stored in non-volatile memory.

2.4.7 Physical Security

The ADS and the proposed cryptocore architecture are designed to make use of PUF features with the aim of minimising the need for physical security. Thus, the only data paths that need to be protected are the raw PUF output and the private key bus (both marked with bold lines in Fig. 4). Apart from these two buses, no other part of the cryptocore requires physical protection, as all other data (in transit or at rest) are considered public. The PUF itself and its internal state are assumed to be protected by the inherent properties of the PUF which make it impossible to perform physical attacks on it without destroying the underlying secret.

Although not strictly required by the scheme, physical and logical isolation of the cryptocore from any other system components provides an additional level of security against physical attacks (i.e. invasive or side-channel attacks).

3 Preliminaries

3.1 Application Scenario

The system comprises a number of networked nodes and at least one authority device which is mobile. The architecture of all the devices, nodes and ADs, is a variant of the reference architecture described in Section 2.2.1 but, to match common Internet-of-Things scenarios, the nodes are assumed to be more resource constrained than the ADs.

In the context of this section, PUFs are considered a component providing the behaviour discussed in Section 2.3 with responses that are reliably reproduced, error-corrected, and entropy-enhanced at the hardware level, since these issues have been extensively studied in literature[3], [24], [29]. Namely, the PUF component can be modelled as an augmented hash function with outputs that are uniformly random based on the internal PUF state and the corresponding challenge.

As per the adversary model discussed below, any data that is stored in non-volatile memory during the operation of the scheme is considered accessible to potential adversaries and should thus be appropriately protected. When, in the remainder of this paper, we refer to storing and retrieving data, we implicitly assume that the integrity of these data is verified. This assumption is based on existing solutions in literature, for example by Hoffman et al.[30].

Symbol	Value or Operation
AD	Authority device
N_x	Node x
P_x	Public key of x
S_x	Private key of x
ID_x	Identifier of device x
$SIG_y(x)$	Signature of data block x with private key y
$VER_y(x, s)$	Verification of the signature s with data x and public key y
$PUF_y(x)$	Evaluation of PUF y with challenge x
$TRNG_y()$	Evaluation of TRNG of device y
$PKG(x)$	Derivation of an asymmetric key pair based on seed x
$\|$	Concatenation
REQ_x	Request to initiate protocol x
ACK	Acknowledgement

Table 1: Summary of symbols

In the basic application scenario for the scheme, a party which we call *the owner*, purchases a number of networked devices (*nodes*). The owner needs to deploy the nodes in the field, creating a network of devices. As in many practical applications, we assume that the nodes are manufactured by an honest entity and then come under the control of the owner who has full authority on them and performs the initial setup. However, for practical or security reasons, the owner might want to delegate the duty of enrolment (performed in the field) to an external party that is again partially trusted. This third party is tasked with enrolling the nodes and given an authority device to make this possible.

The authority device is designed to be connected to each of the nodes. The connection can be wired or wireless but a physical proximity between nodes and ADs is required while they are taking part in a protocol, to ensure the physical validation of the nodes. Upon connection, the AD performs the necessary operations to enable the nodes to act as a group and effectively join the same *ownership domain* or *neighbourhood*. It should be highlighted that none of the configuration requires special expertise from any of the human operators.

Multiple ownership is achieved with different authority devices. Using a large number of ADs does not affect the scalability of the scheme as those devices only take part in operations that are performed offline and not during the normal operation of the system. Also, as evidenced in the prototype implementation (discussed in Section 6), the size of the information that devices have to retain throughout the protocol is relatively small in comparison to modern device capabilities. In any case, most practical scenarios would require the use of a limited number of authority devices. In the following sections we will refer to a maximum of two authority devices for clarity, although all protocols are designed to support any non-zero number of ADs.

The scheme employs the properties of Physical Unclonable Functions to: (a) securely generate cryptographic keys, without the need of keeping them in storage, and (b) enable the system entities to undeniably prove and verify the identity of their interacting parties.

3.2 Notation

To simplify the protocol descriptions, we make use of the notation summarised in Table 1. Operations that are considered well-known or have been described in a different section, are omitted from the descriptions for clarity, unless they are crucial for presentation of the protocol.

3.3 Adversary Model

The various protocols of the Authority Device Scheme can be divided into two broad phases: initialisation, and normal operation. It is assumed that the initialisation phase is performed in a secure environment or using secure channels established via other methods. This assumption means that there are no adversaries involved in the protocols of this phase.

On the other hand, the system spends the majority of its lifetime in normal operation where adversaries can be active. Our adversary model is based on the Dolev-Yao model[31] expanded and modified to include the physical properties provided by unclonability and PUFs. In summary, the following assumptions are made:

Channel: There is no limitation to the physical medium of communication i.e. wired or wireless. The adversary can eavesdrop on any communication without being detected.

Adversary Capabilities: The adversary is able to observe, intercept, modify, delay, replay and synthesise messages. She is able to guess keys and run the cryptographic algorithms involved in the protocols. She is however unable of directly attacking the algorithms themselves.

It is assumed to be computationally impractical, over the lifetime of the system, for the adversary to exhaustively search the cryptographic key space or the CRP space of any PUF chips or otherwise accurately generate new CRPs without access to the chips themselves.

On a physical level, the adversary has the ability to observe the operation of the system and its individual parts either during normal operation or by removing and probing the hardware. In other words, data in device memory and data buses are considered to be available to the adversary unless they are contained in the cryptocore specified in Section 2.4. Most importantly, due to the properties of PUFs, the adversary is unable to probe any PUF chips and extract information that they do not make available through their interfaces, without destroying their internal secrets.

Protocol: The adversary acts as a legitimate node and is capable of initiating and taking part in protocols with any of the parties involved in the scheme.

4 Protocols

The proposed scheme acts as an enabler for the unclonability stack of Section 2.2 through providing the following features:

- Key material is initially generated when the device is powered on using the inherent, unclonable randomness of the PUFs. The key is regenerated when needed without being stored in non-volatile memory. (Protocol 1: Key Generation)
- The ADs and the nodes can be introduced prior to deployment, as an extra layer of security. This allows for a decoupling of the owner and the actual holder of the authority device. (Protocols 2 and 3: Node Setup, Node Verification)
- Nodes can be added to one or more ownership domains (neighbourhoods) by interacting with one or more authority devices. (Protocols 4 and 5: Node Enrolment)
- After their addition to a neighbourhood, nodes are able to become members of additional neighbourhoods with additional authority devices but only after the approval of the initial authority device. (Protocols 5 and 7: Authority Device Authentication, Node Enrolment)
- After their addition to a neighbourhood, nodes are aware of their membership, can exchange public keys with their neighbours, and authenticate them. (Protocols 8 and 9: Node Key Exchange, Node Mutual Authentication)

- After their enrolment, nodes are able to authenticate their owning authority devices. (Protocol 7: Authority Device Authentication)
- Authority devices are able to decommission nodes of their authority domain, effectively removing them from the neighbourhood. (Protocol 6: Node Decommission)

The required set of protocols can be divided in four conceptual domains: key generation, member preparation, membership management, and member interaction. These domains and their relationships are visualised in Fig. 5.

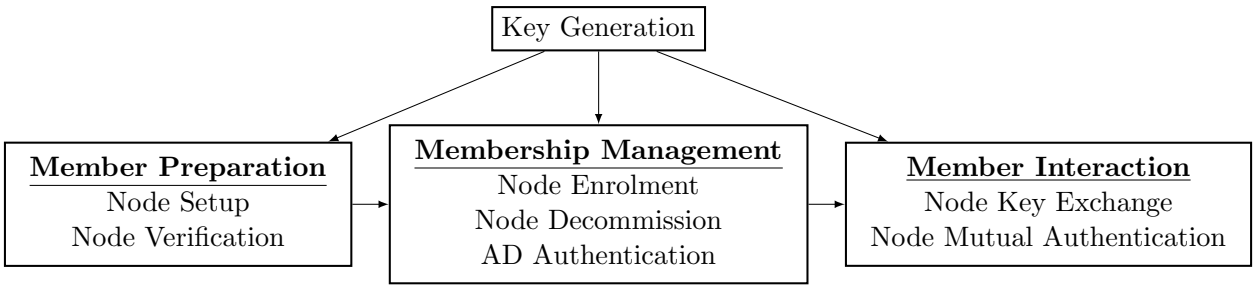
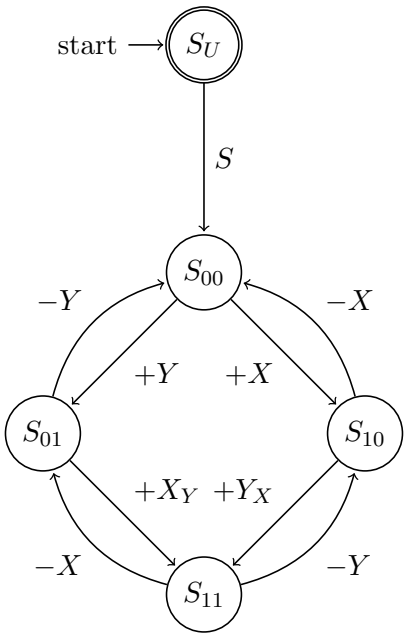


Figure 5: ADS Protocol Domains

To formulate the different protocols we use a multiple ownership scenario, including two authority devices and a number of nodes. Every node starts with no configuration in state S_U. After performing the Setup process (reaching state S_{00}) and being deployed in the field, the node is enrolled by one of the authority devices X or Y and transitions to states S_{10} or S_{01} respectively. Subsequently, the node can start interacting with other enrolled nodes, enrolled with the second AD, or decommissioned. The different states of a node in the two-AD scenario are shown in Fig. 6. As can be seen from the descriptions below, the assumption of only two ADs is merely contributing to the clarity of our descriptions and all methods and protocols can be extended to an arbitrary number of authority devices.



Symbol	Action
S	Node Setup
$+X$	Enrolment with X
$+Y$	Enrolment with Y
$+X_Y$	Enrolment with X, approved by Y
$+Y_X$	Enrolment with Y, approved by X
$-X$	Decommission with X
$-Y$	Decommission with Y

Figure 6: Node states in two-AD scenario with authority devices X and Y

4.1 Key Generation

An important feature of the ADS is the elimination of private data from both non-volatile memory and human interaction. The unclonable key generation process connects the two lower

layers of the unclonability stack, the *unclonable provider* and the *unclonable core*.

The generation takes place on every device after it is powered on, using the entropy provided by the PUF block by challenging it with a random input. The output of the PUF is then used as a seed for the key generation of the chosen public key algorithm.

The random challenge is stored in non-volatile memory to allow the reproduction of the key pair after a device power cycle. To avoid repeating expensive calculations, public keys can also be stored in non-volatile memory and recalled, but their private counterpart is never stored or shared outside the boundary of the cryptocore.

Protocol 1 (Key Generation)**.** *Device D equipped with a cryptocore. At the end of the protocol, D possesses an public key P_D and a secret key S_D.*

1. *D generates random challenge $C = TRNG_D()$.*
2. *D evaluates the PUF with the challenge, generating a seed $R = PUF_D(C)$ and derives the key pair $(P_D, S_D) = PKG(R)$.*
3. *D stores the challenge C to allow key regeneration.*

4.2 Node Setup

Prior to the deployment of the nodes, a preparation step is performed between the AD and the nodes in an environment controlled by the system owner and assumed to be secure. This preparation serves as an introduction for nodes and ADs, configuring them to recognise each other. This is achieved securely, combining the secrets of the node and the AD to protect the CRP of the node in case of the AD is compromised. In addition, this process also enables the important feature of *authority delegation*, since it allows for the AD to be passed to an honest-but-curious third party which will perform the node enrolment after deployment in the field.

Protocol 2 (Node Setup)**.** *Node N and authority device X. At the end of the protocol, N and X have been introduced and are able to verify each other. See Fig. 7.*

1. *X verifies that N has not been decommissioned and aborts on failure.*
2. *X generates a random challenge $C_N = TRNG_X()$ and sends it to N.*
3. *N uses C_N as a challenge to its PUF to generate a response $K_N = PUF_N(C_N)$.*
4. *N sends (K_N, P_N) to X.*
5. *X generates the response $R_N = PUF_X(K_N)$.*
6. *X replies with its public key P_X.*
7. *N stores P_X in its list of potential ADs and replies with an acknowledgement.*
8. *X stores the tuple (P_N, C_N, R_N).*

4.3 Node Verification

The verification protocol is based on the CRP database that was created on the AD during the Setup protocol described above. The protocol essentially makes the AD unable to enrol any nodes other than the ones pre-approved during Setup, under the control of the system owner.

The verification protocol is only designed to be run right after the deployment of the nodes, or in case changes need to be made to the configuration of the system. As a result, to prevent replay attacks, the CRP exchanged during this protocol is discarded and a new Setup protocol execution is required before further verifications.

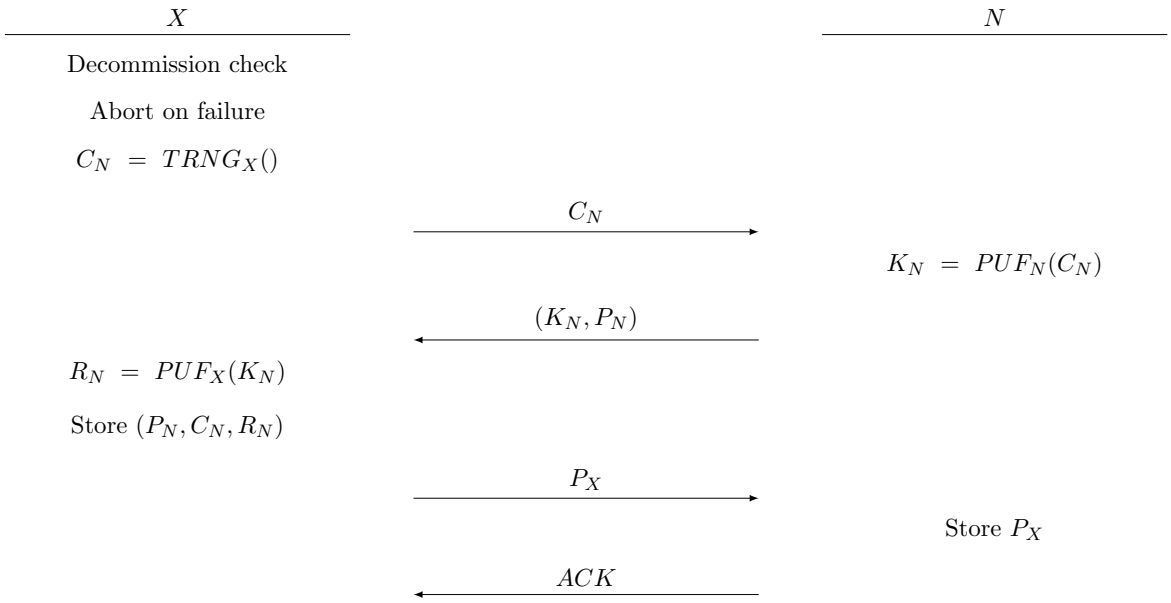


Figure 7: Node Setup

Protocol 3 (Node Verification)**.** *Node N and authority device X. At the end of the protocol, X has verified that it was introduced to N during the Setup protocol. See Fig. 8.*

1. *X sends a verification request to N along with its public key P_X.*
2. *N verifies that P_N exists in the list of authorised ADs created during Setup and aborts on failure.*
3. *N replies with its public key P_N.*
4. *X verifies that N has not been decommissioned and aborts on failure.*
5. *X retrieves the tuple (P_N, C_N, R_N) from its database.*
6. *X signs the challenge C_N and sends the signature along with the challenge to N.*
7. *N verifies the signature and aborts on failure. This prevents unauthorised parties from querying the PUF of N.*
8. *N uses C_N as a challenge to its PUF to generate $K'_N = PUF_N(C_N)$ and sends K'_N to X.*
9. *X generates the PUF response $R'_N = PUF_X(K'_N)$, verifies that $R'_N = R_N$ and aborts on failure.*
10. *X replies with an acknowledgement and discards (C_N, R_N).*

4.4 Node Enrolment

Node enrolment is equivalent to an AD (and its holder) claiming ownership of a node. An enrolled node is regarded as a member of the neighbourhood controlled by the corresponding authority device and possesses the necessary information to prove its membership and communicate with other members. In practice, the AD adds a node to a neighbourhood by signing its public key, thus certifying the validity of the public key.

There are two variants of this protocol to satisfy the scheme goals: (a) for nodes that have not been enrolled before, shown in Protocol 4, and (b) for nodes that are already enrolled, shown in Protocol 5. In the latter case, at least one previous AD is required to participate in

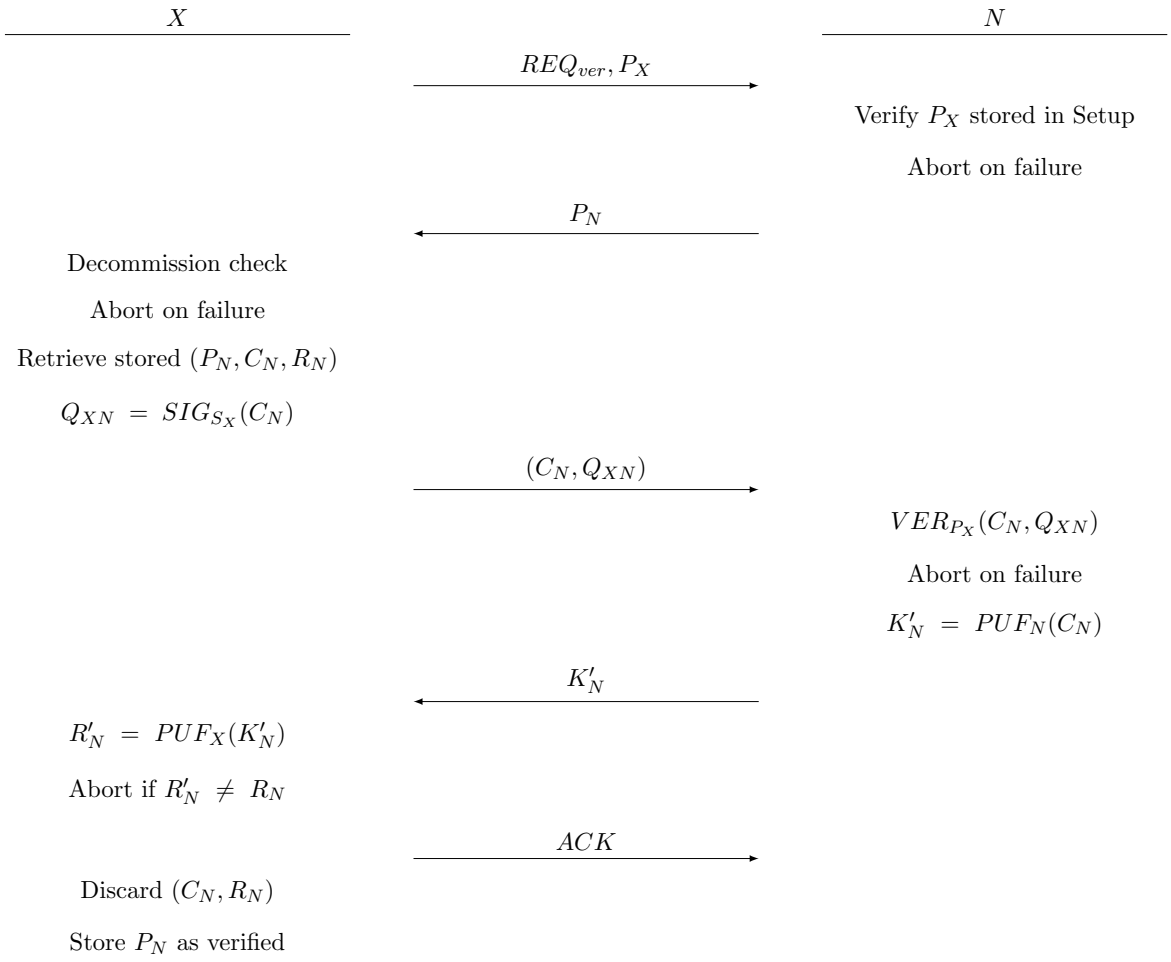


Figure 8: Node Verification

an approval process for the new AD. The task of verifying the legitimacy of the new AD is the responsibility of the holder of the previous AD is thus not included in the protocol. Enrolling with a new AD however, does not remove any previous owners but creates joint ownership relationships.

Protocol 4 (Node Single Enrolment)**.** *Authority device X and Node N, previously set up with X but not enrolled. At the end of the protocol, N is enrolled with X. See Fig. 9.*

1. *X verifies that N has not been decommissioned and aborts on failure.*
2. *X executes a Verification protocol with N and aborts on failure.*
3. *X initiates the enrolment by sending an enrolment request and its public key P_X to N.*
4. *N verifies that P_X is in the list of approved ADs and aborts on failure. This list is populated by either the Setup protocol (Protocol 2) or the first part of the multiple enrolment process (Protocol 5).*
5. *N executes the AD authentication protocol with X (Protocol 7) and aborts on failure.*
6. *N sends P_N to X.*
7. *X verifies P_N against the one stored during the Setup protocol and aborts on failure.*
8. *X generates $Q_{XN} = SIG_{S_X}(P_N)$ and sends (Q_{XN}, P_X) to N.*

9. *N verifies that the received signature matches the received public key and its own public key and aborts on failure.*
10. *N stores (Q_{XN}, P_X) and replies with an acknowledgement.*
11. *X stores N in its list of enrolled nodes.*

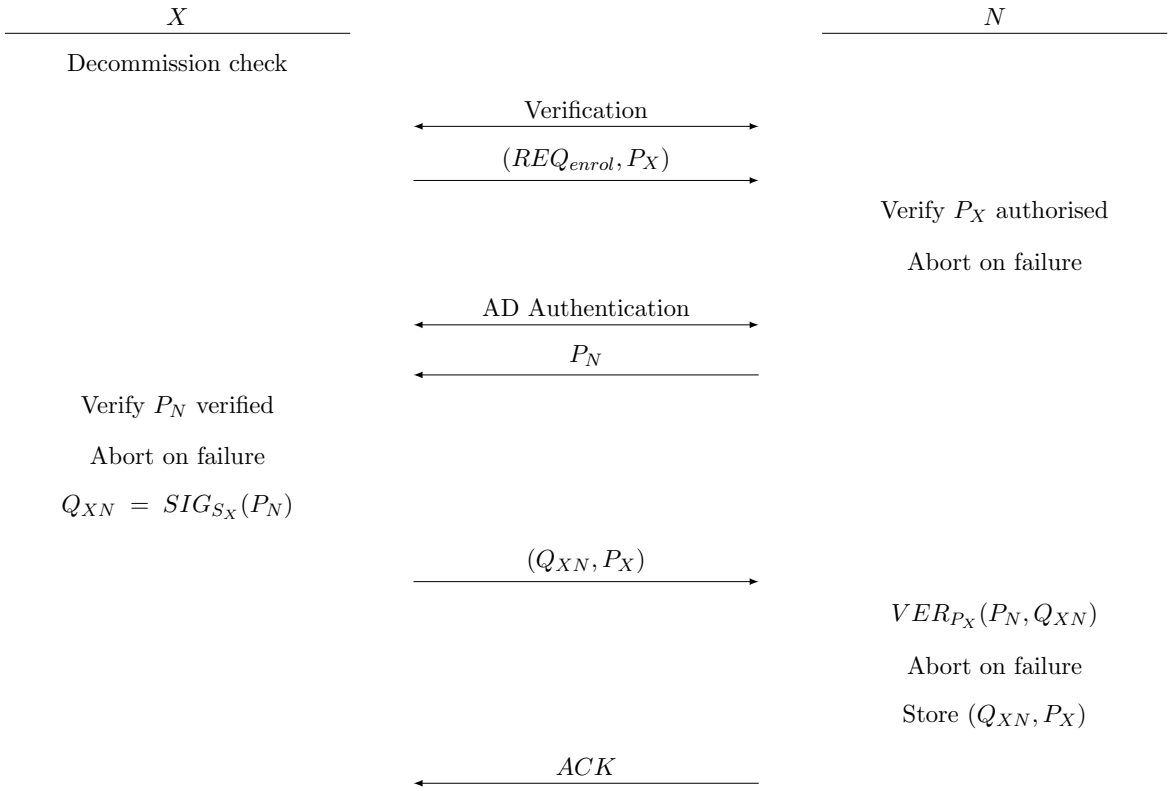


Figure 9: Node Single Enrolment

In the multiple enrolment case, the Verification protocol is omitted, since new, unseen at the time of Setup, ADs might be added over the lifetime of the node. Attacks where an adversary eavesdrops on the communication between Y and N and later replays the

Replay attacks by recording the (P_Y, Q_{XY}) and replaying it for a different, unauthorised AD are implicitly prevented: N initially accepts the new AD (since the signature verification is successful) but in the subsequent enrolment protocol, the malicious AD will have to perform an AD authentication protocol using S_Y. The malicious AD is unable to do that without compromising the private key, which never leaves the boundaries of the cryptocore of Y.

Protocol 5 (Node Multiple Enrolment)**.** *Authority devices X and Y, and node N, previously enrolled with X. At the end of the protocol, N is enrolled with Y and stays enrolled with X. See Fig. 10.*

1. *Y sends ID_N to X.*
2. *X verifies that N is enrolled and aborts on failure.*
3. *X replies with an acknowledgement.*
4. *Y sends P_Y to X.*
5. *X produces a signature $Q_{XY} = SIG_{S_X}(P_Y \| P_N)$ and sends it to Y.*
6. *Y replies to X with an acknowledgement and sends (P_Y, Q_{XY}) to N.*

7. *N verifies the signature with P_X, stored during its enrolment $VER_{P_X}(P_Y \| P_N, Q_{XY})$ and aborts on failure.*
8. *N stores P_Y as an authorised authority device.*
9. *N replies to Y with an acknowledgement.*
10. *Y initiates the enrolment by sending an enrolment request and its public key P_Y to N.*
11. *The remainder of the process is identical to Protocol 4, skipping the node verification step.*

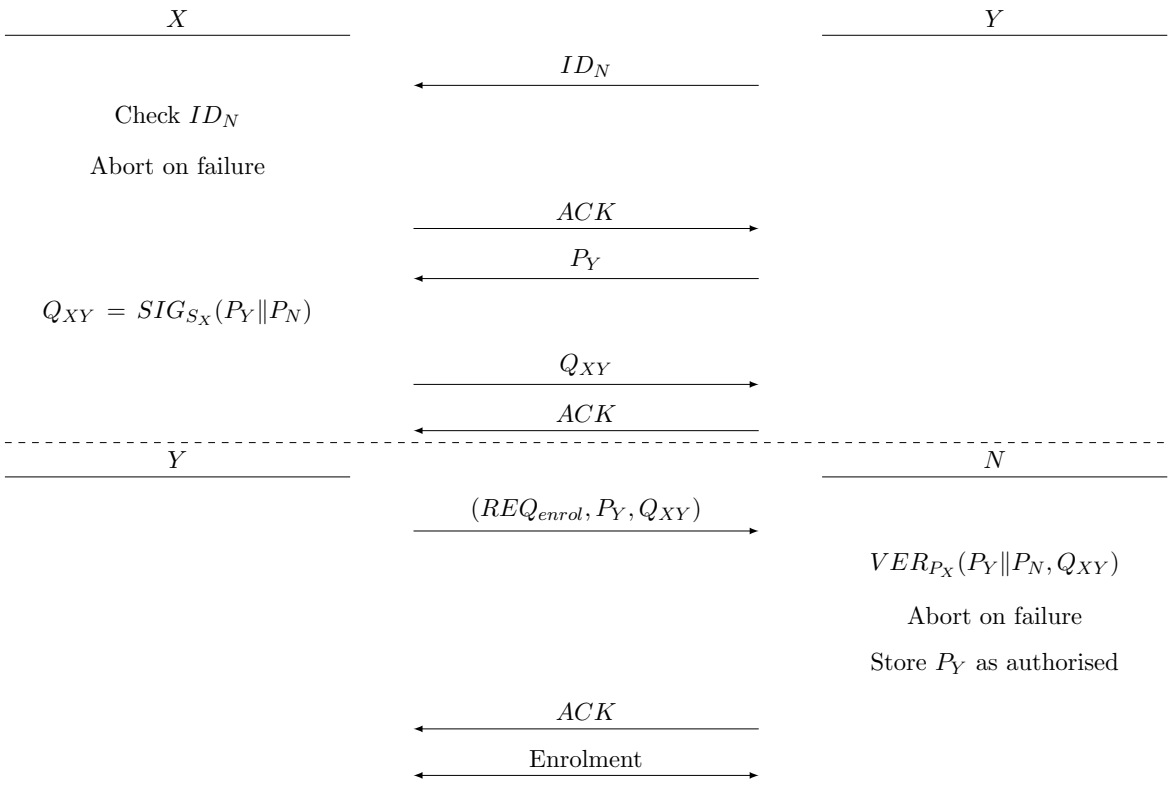


Figure 10: Node Enrolment: Multiple Ownership

4.5 Node Decommission

Decommissioning a node refers to removing it from one of its neighbourhoods. Since the neighbourhood is controlled by an AD, this AD is responsible for instructing the node to delete any signed keys. As mentioned, the nodes are assumed to be visually verifiable during their interactions with an authority device. Additionally, the decommission protocol includes a step of verifying that the node is still in possession of its private key and, in extension, of its PUF. Based on these guarantees, the actual deletion of the signed keys is left to the node, and no further key revocation takes place.

Additionally, ADs keep a 'black list' with the nodes they decommission, in order to take appropriate action if they encounter them again. In case a node is believed to be compromised, a new enrolment round can take place with a fresh key pair for the authority device. At the end of this round, all previous signatures of the AD would be rendered invalid and the nodes would instantly seize to accept them.

Protocol 6 (Node Decommission)**.** *Authority device X and Node N, previously enrolled with X. At the end of the protocol, N no longer belongs to the neighbourhood controlled by X, has erased the relevant identifiers, and has been added to a decommission 'black list' kept by X. See Fig. 11.*

1. *X initiates the protocol by sending a decommission request to N, including its public key P_X as an identifier.*
2. *N executes the AD authentication protocol with X (Protocol 7).*
3. *N removes (Q_{XN}, P_X) from its storage and replies with an acknowledgement.*
4. *X removes N from its list of enrolled nodes and adds it to its black list.*

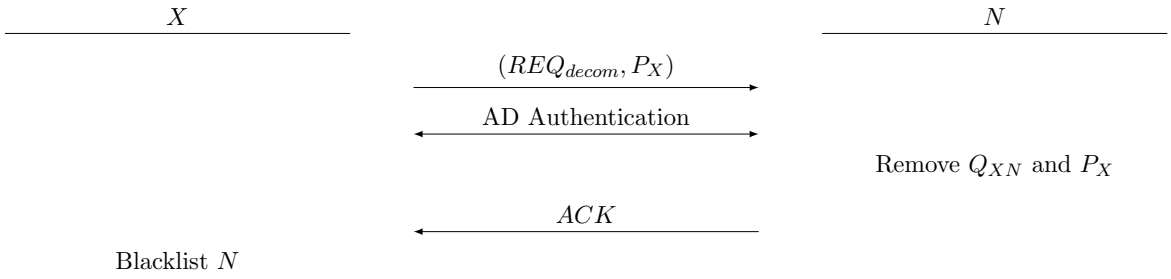


Figure 11: Node Decommission

4.6 Authority Device Authentication

Since the authority device has control of node membership, any subsequent ownership operations need to be performed with the approval of that device. Thus, before accepting any command, the nodes have to authenticate their owning ADs.

Protocol 7 (Authority Device Authentication)**.** *Authority device X and node N, previously enrolled with X. At the end of the protocol, N has authenticated X as one of the ADs with which N has been enrolled. See Fig. 12.*

1. *N generates a random nonce $T = TRNG_N()$ and sends it to X.*
2. *X signs the nonce generating $Q = SIG_{S_X}(T)$ and sends it N.*
3. *N verifies the signature $VER_{P_X}(T, Q)$ and aborts on failure.*
4. *N replies with an acknowledgement.*

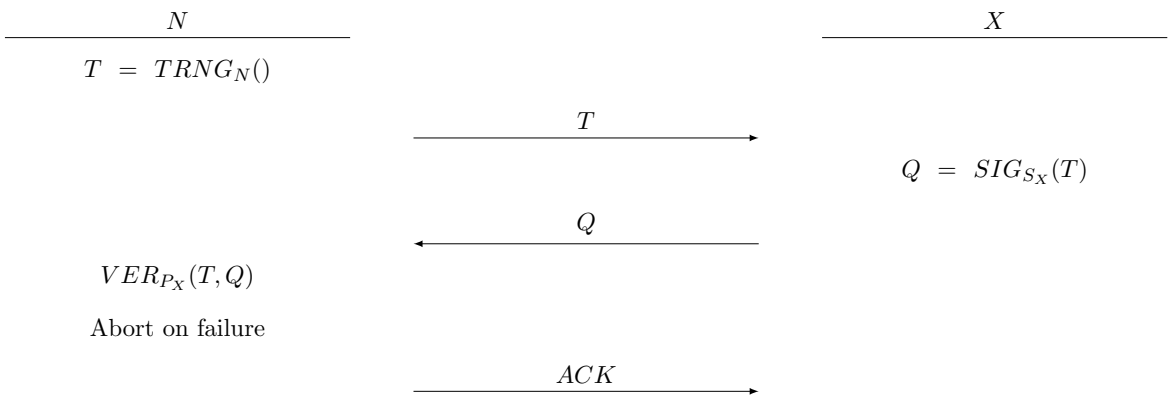


Figure 12: Authority Device Authentication

4.7 Node Key Exchange

This protocol is used to exchange keys between nodes, resembling public key certificate methods. Since each node's public key is signed by the AD during the enrolment phase, all nodes can verify each other's key validity using the public key of the AD. This key, also stored on the nodes while they were enrolled, acts as an *authority anchor*. By verifying the signature of each other's public key, the nodes can identify their neighbours and form neighbourhood relationships that will enable further, higher-level protocols. Additionally, the exchanged keys can be used for the establishment of secure communication channels between the nodes, to provide any required application-level services.

Protocol 8 (Node Key Exchange)**.** *Nodes N_1 and N_2 belonging to the same neighbourhood, both enrolled with AD X. At the end of the protocol, both nodes possess the public keys of each other. See Fig. 13.*

1. *N_1 initiates the protocol by sending the AD public key P_X to N_2.*
2. *N_2 verifies that it has been enrolled with X and aborts on failure.*
3. *N_2 replies with $(ID_{N_2}, P_{N_2}, Q_{XN_2})$ as stored in the enrolment phase.*
4. *N_1 verifies the received key and signature $VER_{P_X}(P_{N_2}, Q_{XN_2})$ and aborts on failure.*
5. *N_1 sends with $(ID_{N_1}, P_{N_1}, Q_{XN_1})$ as stored in the enrolment phase, to N_2.*
6. *N_2 verifies the received key and signature $VER_{P_X}(P_{N_1}, Q_{XN_1})$ and aborts on failure.*
7. *N_2 replies with an acknowledgement.*
8. *If successful, N_1 stores (ID_{N_2}, P_{N_2}) and N_2 stores (ID_{N_1}, P_{N_1}).*

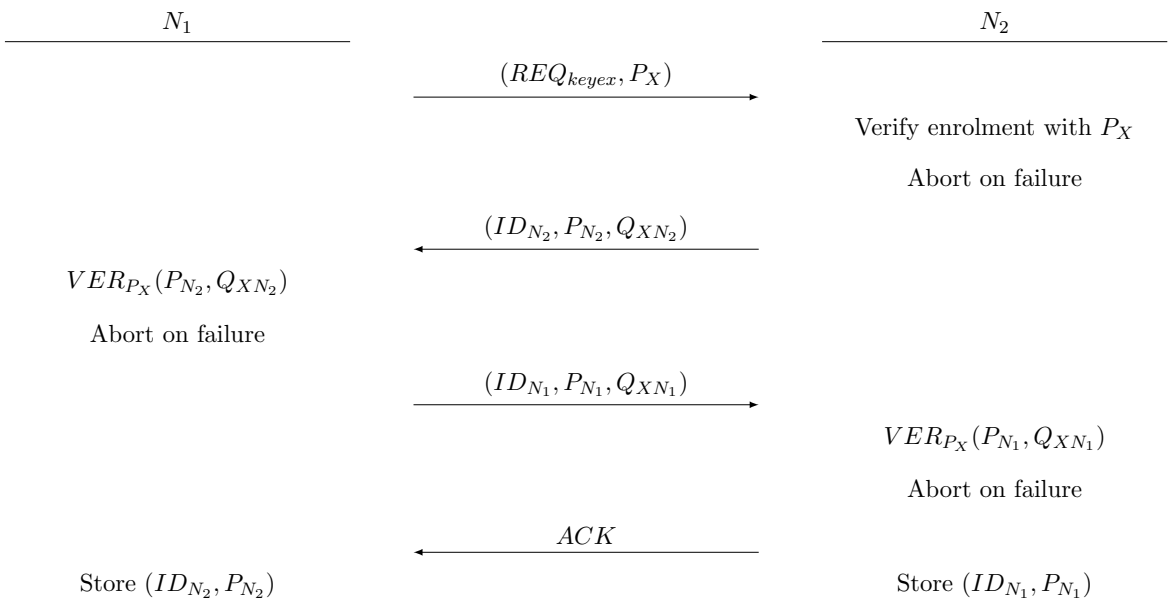


Figure 13: Key Exchange

4.8 Node Mutual Authentication

Based on exchanged public keys, nodes can mutually attest the authenticity of their remote partners by taking turns in signing and verifying a random nonce with their respective key pairs. As with all asymmetric cryptography operations, signing and verifying signatures is

relatively expensive and thus node authentication is designed to be part of relatively infrequent protocols.

Protocol 9 (Node Mutual Authentication)**.** *Nodes N_1 and N_2 belonging to the same neighbourhood, both enrolled with AD X, and have previously exchanged public keys. At the end of the protocol, the nodes have authenticated each other and verified each other's group membership. See Fig. 14.*

1. *N_1 retrieves (ID_{N_2}, P_{N_2}) which was stored during the Key Exchange protocol.*
2. *N_1 generates a random token $T_1 = TRNG_{N_1}()$ and signs it $Q_1 = SIG_{S_{N_1}}(T_1)$.*
3. *N_1 initiates the authentication by sending $(REQ_{auth}, P_X, ID_{N_1}, T_1, Q_1)$ to N_2.*
4. *N_2 verifies that ID_{N_1} is in its list of peers enrolled by P_X and aborts on failure.*
5. *N_2 retrieves (ID_{N_1}, P_{N_1}) which was stored during the Key Exchange protocol.*
6. *N_2 verifies the received signature $VER_{P_{N_1}}(T_1, Q_1)$ and aborts on failure.*
7. *N_2 generates its own random token $T_2 = TRNG_{N_2}()$ and signs it $Q_2 = SIG_{S_{N_2}}(T_1 \| T_2)$.*
8. *N_2 sends the token and the signature to N_1.*
9. *N_1 verifies the received signature $VER_{P_{N_2}}(T_1 \| T_2, Q_2)$ and aborts on failure.*
10. *N_1 replies with an acknowledgement.*

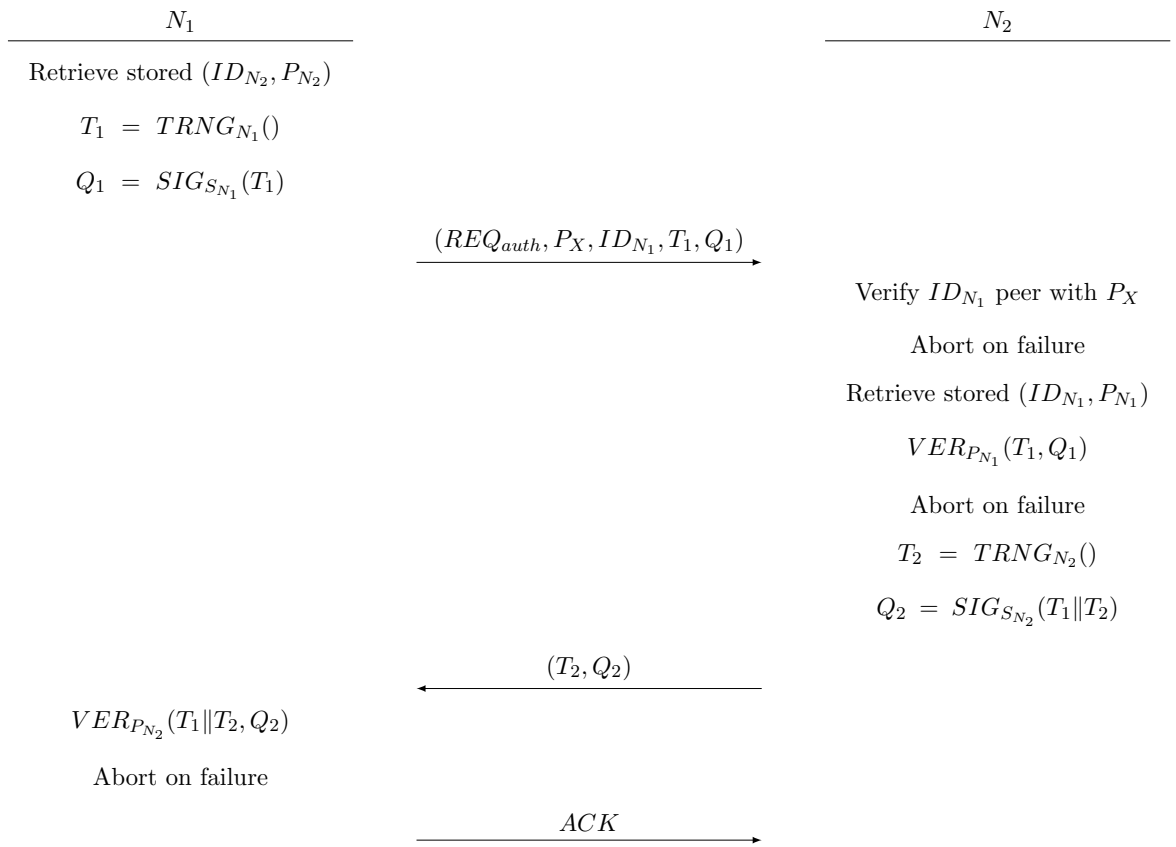


Figure 14: Node Mutual Authentication

5 Security Analysis

In this section we present a security analysis of the Authority Device Scheme in the form of a number of lemmas and theorems. In the following, we are taking into account a single PPT adversary, referred to as $\mathcal{A}$. Since the basis of our hypothesis is the unclonability of the system entities, our analysis focuses on the ability of $\mathcal{A}$ to masquerade as a legitimate entity, be it an AD, an enrolled node, or a new node.

Lemma 1. *The security of the cryptographic algorithms employed in the ADS is guaranteed.*

Proof. Proving the security of the low level cryptographic primitives is out of the scope of our work. As such, we assume that those primitives fulfil their purpose in a secure way. Due to the relaxed requirements for specific cryptographic algorithms, the ADS can be easily adapted to use algorithms proposed in the future as long as they are secure against existential and selective forgery. □

Lemma 2. *In the event of a node compromise, an adversary $\mathcal{A}$ is not able to gain access to or tamper with the internal secrets of the node.*

Proof. Based on the features of the cryptocore discussed in Section 2.4, attempts in invasive probing or tampering with the cryptocore will result in the internal secrets being either destroyed or significantly transformed. Both cases would lead to inability of legitimate operation for the node. □

Lemma 3. *The private key of all entities is physically protected and never exposed outside the boundaries of the cryptocore.*

Proof. As seen in Protocol 1, the key pair for every entity is dynamically generated by its PUF and is never stored in non-volatile memory. Due to the unpredictability property of the PUF, it is infeasible to derive the private key directly from the challenge, without access to the generating PUF chip. In addition, the private key is only used inside the cryptocore which by Lemma 2 is physically secure. □

Lemma 4. *An adversary $\mathcal{A}$ with access to an entity for a short period of time, is not able to copy the entity's PUF.*

Proof. According to the properties of PUFs, as they were introduced in Section 2.3 the following are true:

- The knowledge of a limited number of CRPs does not allow $\mathcal{A}$ to predict additional CRPs.
- The CRP space is sufficiently large and given the PPT capabilities of $\mathcal{A}$, she is unable to exhaustively query the PUF in a bounded time period.
- $\mathcal{A}$ is unable to control the PUF behaviour with the aim of producing specific responses.

Additionally, $\mathcal{A}$ is unable to manipulate the protocols involved in ADS in order to gain access to the internal secrets of any PUF, as shown in Lemma 5 and Lemma 7 and a particularly small number of CPRs are used during the expected lifetime of the system, rendering any modelling attempts impractical. □

Lemma 5. *Upon inspection of the storage of a compromised authority device, an adversary $\mathcal{A}$ is not able to recover the CRPs corresponding to any node.*

Proof. The Setup protocol shown in Protocol 2, makes use of both the PUF of the AD and the PUF of the node. This method essentially combines two internal secrets from the node and the AD into a unique output. Additionally, due to the unpredictability property of PUFs and as discussed in Lemma 4, $\mathcal{A}$ is unable to predict the response of a PUF to a given challenge, except with negligible probability. Thus, the response of the node is stored in a form that provides no sensitive information about the node's PUF in case of a later compromise of the AD. □

Lemma 6. *A legitimate entity can authenticate to another legitimate entity, except for negligible probability.*

Proof. A legitimate entity would have access to its PUF thus being able to generate the required key pair and successfully participate in the authentication protocols of Protocol 7 and Protocol 9. □

Lemma 7. *An eavesdropping or man-in-the-middle adversary $\mathcal{A}$ is not able to use previously transmitted data to successfully perform any of the protocols of the scheme.*

Proof. The data transmitted during the protocols comprises:

- Random tokens are generated by a TRNG. Due to the properties of TRNGs the token values are never repeated and thus replaying of older messages containing a random token will fail.
- Public keys which do not provide any advantage if they are recorded and replayed, since all protocols involve the use of a secret information counterpart to the public keys.
- Signatures generated with secret keys that are not permanently stored and never revealed to third parties.
- PUF CRPs involved in the Setup(Protocol 2) and Verification(Protocol 3) protocols.
- When random token are not included in the interactions, replay attacks are implicitly prevented by the protocols, as discussed in the relevant protocol descriptions.

Evidently, the only truly useful information exchanged between system entities are the PUF CRPs. However, the Setup protocol is performed in a secure environment in the absence of adversaries and the Verification protocol discards used CRPs thus replaying them would have no result for $\mathcal{A}$. □

Lemma 8. *An adversary $\mathcal{A}$ is able to impersonate or clone any authority device or node, only with negligible probability.*

Proof. Before a node has been introduced to an AD with the Setup protocol, it is probable that it is replaced with a malicious node. However, this is deemed impossible due to the secure environment assumption for the Setup protocol.

After the node has performed the Setup protocol, by virtue of 1, impersonating a system entity would require the compromise of said entity's private key, due to Lemma 2. Thus, the probability of the former being impersonated is the same as that of predicting the responses of the node's PUF. By Lemma 4 this probability is negligible. □

Lemma 9. *In the event of a node compromise, an adversary $\mathcal{A}$ does not gain any advantage towards compromising the rest of the system.*

Proof. A node compromise can take place either during its normal operation or after it has been decommissioned. In the first case, the node does not have access to secret information of any entity but itself. At the same time, due to Lemma 2 the security of the node's PUF and secret key is guaranteed. In the second case of compromise after decommission, the above still holds, with the addition of Lemma 10. □

Lemma 10. *An adversary $\mathcal{A}$ is not able to re-enrol a decommissioned node.*

Proof. The authority device corresponding to a neighbourhood is always involved in the enrolment of new nodes in that neighbourhood. In addition, as shown in Protocol 6, the authority devices keep a list of nodes that were previously decommissioned and thus will disallow any re-enrolment attempts. □

Lemma 11. *An adversary $\mathcal{A}$ is not able to use a malicious authority device to gain control the system.*

Proof. During the Enrolment protocol (Protocol 4 and Protocol 5), the nodes verify that the ADs attempting to enrol them are legitimate via a list of authorised ADs. This list is populated during either the Setup protocol or the Enrolment protocol itself, in case of multiple enrolments. The Setup is performed in a secure environment, disallowing the existence of adversaries. In addition, when attempting to enrol a node which has already been enrolled, an AD is required to obtain the approval of at least one of the ADs previously used to enrol the same node, which would be unsuccessful for a malicious AD. □

Theorem 1. *Given the security of the appropriate ADs, there can be no change to the membership attributes of any node, whether the node is legitimate or malicious.*

Proof. By the above lemmas, it is evident that modifications of any form to the membership of a node in any neighbourhood require the participation and authorisation of the appropriate authority devices. Great care has been taken to minimise the trust relationships among the entities taking part in the ADS enabling those ADs to become the roots of trust for the system.

This is an important feature of the scheme since it allows for the verification of the state of the system at any point, as well as the assurance that there are no distortions to the neighbourhood memberships as long as the authority devices is under the control of trusted individuals. Providing further security for those ADs is mainly a social rather than technical issue but, due to the unclonability properties of our scheme, ADs are bound to be unique, existing at only a single place at any given moment, thus guaranteeing the detection of their physical compromise. □

Theorem 2. *The Authority Device Scheme ensures the confidentiality of all private information used in the context of the scheme.*

Proof. The private information used by all entities throughout the operation of the ADS falls under two categories: PUF secrets, and private keys. Both are kept confidential on a physical level as shown in Lemma 2 and are never shared as shown in Lemma 7. Additionally, PUF secrets are protected as per Lemma 4 and Lemma 5, and private keys are unavailable to adversaries as shown in Lemma 3. □

Theorem 3. *The Authority Device Scheme provides a method for the establishment of network clusters, in an unclonable and physically secure manner.*

Proof. Using the above lemmas, we can conclude that the Authority Device Scheme is both correct and secure.

By Theorem 2 only legitimate nodes are able to join and leave neighbourhoods and by Lemmas 6 and 7 only nodes are be able to interact in those neighbourhoods. At the same time, malicious attempts to access and modify the nature of the neighbourhoods are prevented and contained as shown in Lemmas 4 and 7 to 11.

Additionally, by Lemma 2 and Lemma 3 we can conclude that private entity information is protected at a physical level. Finally, by Lemma 5, Lemma 7, and Theorem 2 we can be confident that the unclonability goals of the scheme are achieved. □

6 Prototype

The proposed scheme was implemented as a prototype virtual network, built on the Linux Kernel Virtual Machine (KVM)[32] virtualisation platform, using Python 3[33]. KVM was selected for the ability to be extensively controlled through a command line interface, allowing for automated of our test network.

Additionally, Python made a good candidate for rapid prototyping since it abstracts away from low level details, especially taking into account its wide library support. At the same time, we were able to use pure TCP as the underlying communication mechanism, with a custom message structure. We used the PyCryptodome library[34] to carry out the required cryptographic operations, due to its wide acceptance in the Python community.

Each message carries a header besides its payload, including the current 'phase' and 'command' of the scheme as well as auxiliary flags and length information, detailed in Fig. 15. The auxiliary flags comprise an initialisation (INIT / I) and final (FIN / F) flag representing the first and last steps of the current protocol respectively. Accordingly, the communicating parties make use of the header information to advance their state and detect potential violations of the protocol. A summary of the header values used in our prototype is given in Table 2.

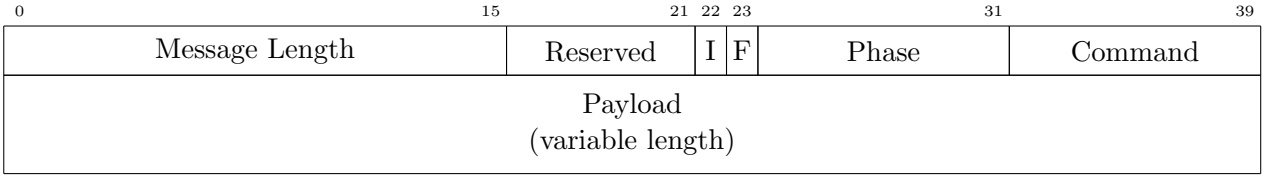


Figure 15: Message structure

Header	**Value**	**Hexadecimal Value**
Phase	Null	00
	Setup	01
	Verification	02
	Enrolment	03
	Decommission	04
	Authentication	05
	Key Exchange	06
	Authority Device Authorisation	07
	Ratchet Setup	08
	Ratchet Step	09
	Ratchet ZK Setup	0A
	Ratchet ZK Step	0B
Command	Null	00
	PUF Challenge	01
	PUF Response	02
	Public Key	03
	Acknowledgement	04
	Failure	05
	Initiate	06
	Signature	07
	Nonce	08
	Id	09
	Commitment	0A
	Proof	0B

Table 2: Message header values

6.1 Implementation Parameters

The aim of this prototype is to verify, as well as showcase the operation of the Authority Device Scheme. Indeed, during the development of the prototype, numerous shortcomings of the protocols became apparent and were rectified in subsequent iterations. As such, the choices of specific implementation parameters, summarised in Table 3, were made based on security guidelines (e.g. [35]) with the aim of creating a close analogy to real world applications, rather than optimising the performance of the scheme.

For the signature operations the Elliptic Curve Digital Signature Algorithm (ECDSA)[36]. In the context of IoT, an Elliptic Curve algorithm is preferable to, e.g. RSA, since it is generally both faster and requires a much smaller key size to achieve the same security level. As an example, for our chosen parameters the RSA key would be 30 times larger than the ECC key for the same level of security.

As seen in the relevant NIST guidelines[35], private keys of 512-bit length for ECC provide the equivalent of 256 bits of security and would be sufficient against the vast majority of adversaries, at least over the next two decades. As a result, we chose to use the NIST P-256 curve which is also widely used in practice.

The cryptographic keys are represented in binary DER format as specified in [37]. This format includes additional information about the algorithm, leading to a slightly larger total size for the key representation: 65 bytes in uncompressed form and 59 bytes in compressed form. Evidently, the compressed form requires less storage space and less bandwidth but also involves the cost of decompression operation every time the key is used. Due to the nature of the ADS as an enabler for other protocols, the frequency of cryptographic operations involving public keys is ordinarily low, and as a result, we used the compressed key form to reduce the required storage space on the devices.

We emulated the PUF component with a randomly generated lookup table to provide the CRP mapping. The internal state of the PUF is seeded by a combination of parameters, aiming to emulate the uniqueness of the component between different instances, despite it being created from the same Python class. Additionally, random bit errors of up to 20% were added to the PUF responses, as this a common maximum error rate in practical PUF instantiations[3].

Since the bit error rate present in the PUF responses was up 20% (or 51.2 bits per 256-bit response), a [n = 255,k =21,t = 55] BCH block code was used for error correction, allowing the removal of up to 55 bit errors while producing 234 bits of helper data.

Object	**Length in bits**
BCH Helper Data	234
PUF Response	256
Random Nonce	256
ECDSA Public Key (Compressed DER)	472
ECDSA Signature	512
Device ID	16

Table 3: Implementation parameters

An overview of the storage requirements for ADs and nodes is given in Table 4. As expected, the storage requirements for the authority devices scale linearly ($O(n)$) with the number of nodes and similarly, peer information stored by nodes themselves is also proportional to the number of peers. There is also a constant cost for each CRP produced by the PUF, since the associated helper data is stored for future use. However, in typical scenarios each node should only be required to produce only one CRP per AD, in addition to the key CRP. Thus the total storage on any device would be in the region of a few hundred kilobytes.

Device Type	Object	Length in bits
All	PUF challenge for the key seed	256
	Helper data for each PUF CRP	234
AD	Public key for each enrolled node	472
	CRP for each node that has been set up	512
Node	Public key for each authorised AD	472
	Public key for each enrolling AD	472
	Signature for own public key from each enrolling AD	512
	Interacting peer public key	472

Table 4: Storage requirements

6.2 Simulation Scenarios

We devised a number of example scenarios to verify and showcase the features of the scheme:

Simple: The basic protocols of setup, verification, enrolment, and decommission are performed between a single node and a single authority device.

Multiple nodes: Takes place in two phases, with two nodes and a single AD. In the first phase, the AD performs the setup, verification, and enrolment operations with both nodes, the equivalent of adding the nodes to a neighbourhood. Subsequently, the nodes take part in a key exchange and take turns authenticating each other.

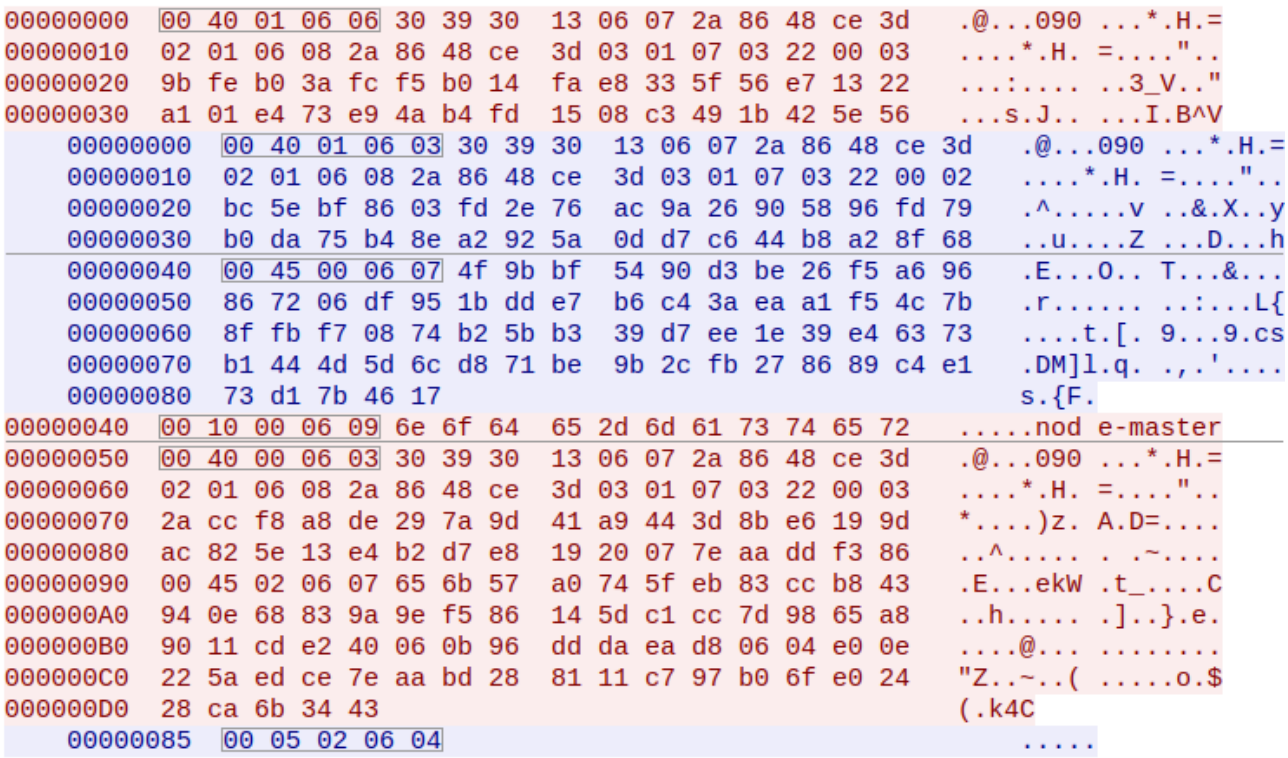
Multiple authority devices: With two ADs and a single node, this scenario demonstrates the interactions involved in managing multiple ownership. The first authority device initially executes the setup and the enrolment with the node and afterwards the second AD proceeds to with the enrolment of the node, after it has received authorisation from the first AD. After the completion of this process, the node reaches a state where it simultaneously belongs to two neighbourhoods.

All scenarios were executed on a virtual network which included a different virtual machine to represent each of the devices. In addition, the TShark network protocol analyser[38] was used to observe the traffic between the virtual machines and verify the validity of the scheme(Fig. 16).

6.3 Observations

The message exchanges among the devices taking part in the above scenarios were analysed in Wireshark[39]. As expected, the high-level functionality of the protocols was confirmed by the results. Nevertheless, a number of implementation issues arose while constructing the prototype leading to further iterations of the scheme, and verifying our initial intuition for the value of a prototype.

Fig. 16 presents an example TCP flow for the Key Exchange protocol between two nodes. In this exchange the nodes have been given the identifiers 'node-master' and 'node-slave' to signify the entity which initiates the protocol. The message headers, marked with a rectangle comprise: the total length, flags, phase, command, and payload. As per Table 2, the value of the Phase header is 0x06 for the all the messages of the Key Exchange protocol. The rest of the headers are detailed in Fig. 16 where their value is given in parentheses following their hexadecimal representation.



Packet	Source	Destination	Length	Flags	Command
1	Master	Slave	0040 (64)	01 (INIT)	06 (Initiate)
2	Slave	Master	0040 (64)	01 (INIT)	03 (Public Key)
3	Slave	Master	0045 (69)	00	07 (Signature)
4	Master	Slave	0010 (16)	00	09 (ID)
5	Master	Slave	0040 (64)	00	03 (Public Key)
6	Slave	Master	0005 (5)	02 (FIN)	04 (Acknowledgement)

Figure 16: TCP packet flow for the Key Exchange protocol

7 Conclusions

Integrating the unclonability primitive in consumer devices would have a profound impact on the societal concepts of ownership, authority, and eventually trust. We have presented ADS, a collection of security protocols that allow the formation of network neighbourhoods, with the aim of exploiting the primitive of unclonability to provide novel features for networked systems, focused on Machine-to-Machine and Internet-of-Things scenarios.

Our work aims to integrate unclonability in the basis of modern networking scenarios while retaining and even improving the usability of existing security solutions. This is achieved in two ways: by designing protocols which guarantee their security without requiring excessive configuration, and by constraining the system authority operations to physical entities that can be secured and verified. The most important advantage of our solution is the combination of these two methodologies into a scheme that isolated the secrets of all the devices of the system, irrevocably locking them into the physical domain, a feature that enables the users to apply, hand over, verify, and destroy the secrets as they would with a physical key. At the same time, this key cannot be duplicated and thus bears a much higher value than it otherwise would.

Our overarching goal is to continue with this integration on multiple networking levels to create a truly unclonable networking stack as it was described in the introduction of this report. To that end, we are already working on constructing methods and protocols which create unclonable link and neighbourhoods, expanding the reach of the unclonability primitive.

In addition, it would be interesting to explore the integration of unclonability and inherent randomness in routing decisions, taking advantage of the highly redundant paths of Machine-to-Machine systems.

References

[1] M.-D. D. M. Yu, M. Hiller, J. Delvaux, R. Sowell, S. Devadas and I. Verbauwhede, 'A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication', *IEEE Transactions on Multi-Scale Computing Systems*, vol. 7766, no. c, pp. 1–1, 2016.

[2] S. Kerr, M. S. Kirkpatrick and E. Bertino, 'PEAR', in *3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*, 2010, p. 18.

[3] R. Maes, *Physically unclonable functions: Constructions, properties and applications*. Springer Berlin Heidelberg, 2013, pp. 1–185, ISBN: 9783642413957.

[4] R. Pappu, 'Physical One-Way Functions', *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[5] J. Guajardo, B. Škorić, P. Tuyls, S. S. Kumar, T. Bel, A. H. M. Blom and G. J. Schrijen, 'Anti-counterfeiting, key distribution, and key storage in an ambient world via physical unclonable functions', *Information Systems Frontiers*, vol. 11, no. 1, pp. 19–41, 2009.

[6] J. Lee, D. L. D. Lim, B. Gassend, G. Suh, M. V. Dijk and S. Devadas, 'A technique to build a secret key in integrated circuits for identification and authentication applications', *Symposium on VLSI Circuits*, pp. 176–179, 2004.

[7] G. E. Suh and S. Devadas, 'Physical Unclonable Functions for Device Authentication and Secret Key Generation', in *44th ACM/IEEE Design Automation Conference*, 2007, pp. 9–14.

[8] D. Mukhopadhyay, 'PUFs as Promising Tools for Security in Internet of Things', *IEEE Design & Test*, vol. 33, no. 3, pp. 103–115, 2016.

[9] B. Gassend, D. Clarke, M. van Dijk and S. Devadas, 'Silicon physical random functions', *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS2002*, no. November, p. 148, 2002.

[10] M. Delavar, S. Mirzakuchaki and J. Mohajeri, 'PUF-based solutions for secure communication in Advanced Metering Infrastructure (AMI)', 2016.

[11] P. Tuyls and B. Škorić, 'Strong Authentication with Physical Unclonable Functions', in *Security, Privacy, and Trust in Modern Data Management*, 2007, pp. 133–148.

[12] K. Goutsos, 'Physical Unclonable Functions in Cryptoblocks', Master's thesis, School of Electrical and Electronic Engineering, Newcastle University, 2015.

[13] J. Guajardo, S. S. Kumar, G.-j. Schrijen and P. Tuyls, 'FPGA Intrinsic PUFs and Their Use for IP Protection', *Lect. Notes Comput Sc.*, vol. 4727, pp. 63–80, 2007.

[14] S. S. Kumar, J. Guajardo, R. Maes, G. J. Schrijen and P. Tuyls, 'The Butterfly PUF protecting IP on every FPGA', in *2008 ru, HOST*, 2008, pp. 67–70.

[15] P. Tuyls, G.-J. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh and R. Wolters, 'Read-Proof Hardware from Protective Coatings', in *Cryptographic Hardware and Embedded Systems*, 2006, pp. 369–383.

[16] U. Rührmair, H. Busch and S. Katzenbeisser, 'Strong PUFs: Models, Constructions, and Security Proofs', in, 2010, pp. 79–96.

[17] K. Kursawe, A.-R. Sadeghi, D. Schellekens, B. Skoric and P. Tuyls, 'Reconfigurable Physical Unclonable Functions - Enabling technology for tamper-resistant storage', in *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, 2009, pp. 22–29.

[18] S. Katzenbeisser, Ü. Kocabaş, V. van der Leest, A.-R. Sadeghi, G.-J. Schrijen and C. Wachsmann, 'Recyclable PUFs: logically reconfigurable PUFs', *Journal of Cryptographic Engineering*, vol. 1, no. 3, pp. 177–186, 2011.

[19] U. Rührmair, C. Jaeger and M. Algasinger, 'An attack on PUF-based session key exchange and a hardware-based countermeasure: Erasable PUFs', *Lect. Notes Comput Sc.*, vol. 7035 LNCS, pp. 190–204, 2012.

[20] S. Meguerdichian and M. Potkonjak, 'Matched public PUF: Ultra low energy security platform', in *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2011, pp. 45–50.

[21] Ł. Krzywiecki, 'Anonymous Authentication Scheme Based on PUF', in *LNCS*, 2016, pp. 359–372.

[22] S. Devadas, E. Suh, S. Paral, R. Sowell, T. Ziola and V. Khandelwal, 'Design and Implementation of PUF-Based "Unclonable" RFID ICs for Anti-Counterfeiting and Security Applications', in *2008 IEEE International Conference on RFID*, 2008, pp. 58–64.

[23] C. Huth, J. Zibuschka, P. Duplys and T. Güneysu, 'Securing systems on the Internet of Things via physical properties of devices and communications', in *SysCon 2015*, 2015, pp. 8–13.

[24] A. Van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A. R. Sadeghi, I. Verbauwhede and C. Wachsmann, 'Reverse fuzzy extractors: Enabling lightweight mutual authentication for PUF-enabled RFIDs', in *Lect. Notes Comput Sc.*, vol. 7397, 2012, pp. 374–389.

[25] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach and S. Devadas, 'Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching', in *2012 IEEE Symposium on Security and Privacy Workshops*, 2012, pp. 33–44.

[26] U. Rührmair, 'SIMPL systems, or: Can we design cryptographic hardware without secret key information?', in *Lect. Notes Comput Sc.*, vol. 6543 LNCS, 2011, pp. 26–45.

[27] Y. Dodis, R. Ostrovsky, L. Reyzin and A. Smith, 'Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data', *SIAM Journal on Computing*, vol. 38, no. 1, pp. 97–139, 2008.

[28] E. Leobandung, *SRAM as Random Number Generator*, 2017. [Online]. Available: https://patents.google.com/patent/US20190182054A1/en.

[29] C. Herder, B. Fuller, M. van Dijk and S. Devadas, 'Public Key Cryptosystems with Noisy Secret Keys', *IACR Cryptology ePrint Archive*, 2017.

[30] C. Hoffman, M. Cortes, D. F. Aranha and G. Araujo, 'Computer security by hardware-intrinsic authentication', in *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2015, pp. 143–152.

[31] D. Dolev and A. Yao, 'On the security of public key protocols', *IEEE Trans. Inf. Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[32] *KVM*. [Online]. Available: https://www.linux-kvm.org/ (visited on 18/03/2019).

[33] *Python.org*. [Online]. Available: https://www.python.org/ (visited on 18/03/2019).

[34] PyCryptodome, *PyCryptodome*. [Online]. Available: https://www.pycryptodome.org/ (visited on 18/03/2019).

[35] E. Barker, 'NIST Recommendation for Key Management Part 1: General', National Institute of Standards and Technology, Tech. Rep., 2016. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf.

[36] D. Johnson, A. Menezes and S. Vanstone, 'The Elliptic Curve Digital Signature Algorithm (ECDSA)', *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.

[37] S. Turner, D. Brown, K. Yiu, R. Housley and T. Polk, *Elliptic Curve Cryptography Subject Public Key Information*, RFC 5480 (Proposed Standard), RFC, 2009. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5480.txt.

[38] *Tshark*. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html (visited on 18/03/2019).

[39] *Wireshark*. [Online]. Available: https://www.wireshark.org/ (visited on 26/03/2019).