18th International Conference on Space Operations, Montreal, Canada, 26 - 30 May 2025.

SpaceOps-2025, ID # 108

DTN Key Management in a Security Context Agnostic Way

Matthes Würbs[a], Lars Baumgärtner [a*], Marcus Wallum [a]

[a] *Ground Systems Engineering and Innovation Department, ESA/ESOC, Darmstadt, Germany,*
m.wuerbs@gmail.com
lars.baumgaertner@esa.int
marcus.wallum@esa.int

* Corresponding Author

Abstract

In the modern internet, cryptography is vital for secure communications and data exchange. BPSec is a security extension for the Bundle Protocol (BP) version 7, a protocol designed for Delay-/Disruption-Tolerant Networks (DTNs) and a foreseen foundation of the upcoming Solar System Internet. BPSec provides cryptographic-based security properties such as confidentiality, integrity or authenticity. At the moment, two default security contexts using symmetric encryption are specified for use with BPSec. They do not, however, specify how these keys should be distributed between participating nodes.

Therefore, BPSec needs a mechanism to distribute and manage keys in DTNs. Unlike in the terrestrial internet, it is not feasible to request keys just-in-time or to perform handshakes requiring multiple round trips, such as those associated with Transport Layer Security (TLS). This is due to the unique challenges posed by DTNs in interplanetary environments including, i.a., high delays, frequent disruptions, lost and out-of-order bundles.

This paper outlines a novel, experimental approach to key distribution in DTNs, leveraging existing BP infrastructure for the routing and forwarding of messages. Our hybrid approach distributes public keys throughout the DTN before they are needed, alleviating the need for interactive communication or handshakes. The distributed public keys can then be used by each node to derive symmetric keys for use with BPSec. Further, the approach allows for cooperation between multiple key authorities or service providers, supports revocations and can utilize multicast. This paper presents the design and prototype implementation of the approach and evaluates its performance through a case study. Whilst not yet subjected to formal security proofs, and within the constraints of standardised default security contexts, results indicate that the scheme can achieve secure and efficient key distribution for DTNs using BPsec, remaining flexible to possible future security contexts and scenarios, ranging from earth observation to lunar and mars communication.

Keywords: communication, security, key-distribution, dtn, bundle protocol, bpsec

Acronyms/Abbreviations

Automated Certificate Management Environment (ACME)
Bundle Protocol (BP)
Bundle Protocol Agent (BPA)
Bundle Protocol Key Distribution (BPKD)
Bundle Protocol Security (BPSec)
Consultative Committee for Space Data Systems (CCSDS)
CBOR Signing and Encryption (COSE)
Delay/Disruption-Tolerant Network (DTN)
Elliptic-curve Diffie-Hellman (ECDH)
Key Derivation Function (KDF)
Mission Control Center (MCC)
Post-compromise Security (PCS)
Perfect Forward Secrecy (PFS)
Relay Control Centre (RCC)
Round-Trip Time (RTT)
Space Data Link Security (SDLS)
Transport Layer Security (TLS)

1. Introduction

The Bundle Protocol (BP) [1] is a network protocol designed to cope with the challenges in Delay/Disruption-Tolerant Networks (DTN) [7]. It is a requirement for future space missions and a critical building block of the future Solar System Internet (SSI) [30], having been referenced in the *Future Lunar* and *Mars Communications Architecture* [2][3] and the *LunaNet Interoperability Specification* [4]. As a security extension for BP, BPSec has been specified in RFC 9172 [5]. It defines data integrity and confidentiality services for BP. The actual cryptographic operations and algorithms are then specified in so called BPSec security contexts.

For cryptographic operations, encryption keys are required. To our knowledge, there are no existing, approved specifications addressing the management of cryptographic keys in the context of the BP. At the time of writing, BPSec relies on pre-shared symmetric keys as specified in the default security context (RFC 9173 [26]). With the growing numbers of nodes, this is however likely impractical and neither future proof nor scalable.

Due to the unique challenges in DTNs, especially in the SSI context, terrestrial approaches to key management are considered unsuitable for key management in space. Hence, a new approach is needed.

1.1 Contributions

The main outcome of this work is a prototype design and implementation of an approach for key management in non-terrestrial DTNs with support for different cipher suites, taking compatibility with the default security contexts as a constraint. The work results in the following contributions:

- A BPSec security context-independent design for key management and distribution in DTNs,
- A proof-of-concept implementation based on ESA's BP implementation,
- Multiple test cases for key management, used for an evaluation in two different scenarios.

1.2 Outline

The next chapter contains an analysis of the challenges in non-terrestrial DTNs as well as requirements for a delay-tolerant key management architecture, derived from those challenges. In chapter 3, we look at related work and the current state of the art in the space industry. Chapter 4 describes the derived design and 5 contains an evaluation using six test cases in two scenarios. A brief conclusion as well as an outlook is given in chapter 6.

2. Problem Analysis & Requirements

This section provides an overview of the main challenges inherent to DTN and space communication, followed by a description of the requirements on different aspects of the architecture.

2.1 Challenges

As mentioned previously, there are many challenges inherent to DTN complicating communications, especially when deployed in an interplanetary environment. This section describes these challenges and their consequences.

CH1 **Delays**
Vast distances in space can lead to long delays in message propagation. While this may only be a few seconds in case of the Moon, the propagation delays to Mars can vary between approximately 4 minutes and up to 24 minutes. In particular when assuming maximum possible delays, this makes it impractical to quickly retrieve encryption keys or to verify certificate or key validity with a terrestrial central authority.

CH2 **Disruptions**
Sometimes, nodes may lose line of sight to other nodes, for instance occultations when they move behind a celestial body like the Moon. This can lead to disruptions which may make retransmission of bundles necessary. Depending on the length of the disruption, it might be necessary to use one or more intermediary nodes to relay bundles, potentially increasing delays further.

CH3 **Round-Trip Times (RTTs)**
Long propagation delays lead to even longer RTTs. Furthermore, there might be times where a round trip is impossible or where links only work unidirectionally. This, again, makes it unrealistic to contact

a central authority to retrieve or validate keys or certificates. Additionally, it may make the establishment of a secure channel or session key through one or more round-trips infeasible.

CH4 **Lost Bundles**
The aforementioned properties can lead to bundles being lost in transmission or exceeding their bundle lifetime when stored too long on a waypoint node, never reaching their destination. For key management, this could lead to missed keys, which is particularly problematic if a node fails to update its own key in time – i.e. before its current keys expire. It might also mean handshakes requiring multiple round-trips may fail and need to be retried.

CH5 **Fragmented Bundles**
Larger bundles may be fragmented into smaller bundles to be sent separately. This may lead to bundles being received only partially, because one part is delayed or lost while another part has already been received.

CH6 **Out-Of-Order Bundles**
Due to the previous circumstances, it is not unrealistic for bundles to arrive in a different order than the order they have been sent in. This could, for example, stem from later bundles having a faster link or a direct line of sight to the destination available. If bundles arrive out of order without precautions taken, handshake sequences could be impacted, or newer data overwritten with older data.

CH7 **Keys Expiring Before Bundle Reception**
Some of the previous challenges may cause a bundle to be received too late, rendering the cryptographic material used by the sender invalid. This can result in otherwise valid bundles being rejected and needing to be resent.

2.2 Requirements

The following subsections describe the requirements for a key distribution architecture able to handle the previously described challenges while still being compatible with the default security contexts. The requirements are split into three categories: *Key Material*, describing the keys and the associated security requirements; *Key Distribution*, describing the key distribution and the supporting communication channel; *Functionality*, describing the system's functionality to properly and securely facilitate key management in DTNs.

2.2.1 Key Material

REQ1.1 **System Must Be Compatible with The Default Security Contexts**

To prevent being blocked waiting for new security contexts to be specified, the solution must be compatible with the already existing default security context.

REQ1.2 **Keys Must Be Asymmetric**

Asymmetric keys simplify key management in two ways: secret keys do not need to be sent through possibly insecure networks, and the total number of keys that need to be distributed is lower than when using symmetric cryptography. This is especially important when considering the increased number of nodes in the future.

REQ1.3 **Cryptoperiods of Keys Must Take Delivery Delays into Account** [CH1]

For security reasons, keys must not be valid indefinitely but instead be assigned a limited lifetime. Due to the nature of DTNs, bundles might arrive delayed. Thus, the originator-usage period and the recipient-usage period may be very different.

REQ1.4 **Must Allow Multiple Keys Per Node** [CH7]

To properly handle CH7, nodes need multiple keys which are valid for different lifetimes or endpoints on the same bundle protocol agent.

2.2.2 Key Distribution

REQ2.1 **Must Distribute Keys and Relevant Metadata to All Participating Nodes**
The system must distribute keys and relevant metadata to all participating nodes. The relevant metadata must at least include lifetime and identity bind information, e.g. in form of a certificate.

REQ2.2 **Key Distribution Must Not Require Manual Interaction**
The system must be able to keep keys on every participating node up to date on its own, requiring human interaction only in edge cases like revocations.

REQ2.3 **Key Distribution Must Happen Before Keys Are Needed** [CH1, CH3]
If keys are not available when needed, they need to be requested interactively. Because of the previously named challenges, this could cause significant delays in message transmission.

REQ2.4 **Communication Must Not Require a Handshake** [CH1, CH2, CH3, CH6]
Since a handshake would require at least one round trip, delaying message transmission, handshaking must be avoided.

REQ2.5 **Key Distribution Must Be Able to Utilize Multicast**
Especially for larger scale networks, multicast could reduce the message overhead as messages would not need to be sent to every node individually.

2.2.3 Functionality

REQ3.1 **Must Allow Revocations**
To quickly invalidate compromised keys, the system must allow revocations.

REQ3.2 **Must Allow Adding New Nodes to The System**
Since new spacecraft, rovers and satellites are launched regularly, it must be possible to add these to the system at any time.

REQ3.3 **Must Allow Nodes to Update Their Key-Pair**
Nodes must be able to update their key-pair in order to maintain valid keys once older keys expire. Nodes must be able to do it autonomously to avoid manual intervention, as stated in REQ2.2.

REQ3.4 **Must Have A Central Key Authority per User Group**
Nodes must not accept keys from untrusted nodes, as this would allow attackers to provide fake keys. A central instance (i.e. a key authority) within a user group offers more precise control over network activities while giving all associated nodes a central source of trust. Further, it also acts as a single source of truth, so no two nodes have conflicting information.

REQ3.5 **Must Support Key Exchange Between Multiple Key Authorities**
While all nodes receive keys from their associated key authority, they must also be able to receive keys from other trusted key authorities to facilitate multi-agency missions and general cooperation.

REQ3.6 **Cryptographic Algorithms Must Be Exchangeable**
Considering post-quantum cryptography, it makes sense to keep algorithms exchangeable to allow future adaptions (often referred to as ‘crypto agility’). This also makes things easier for implementers with different cryptographic standards to still use the same key distribution mechanisms for their key material.

3. Related Work

In this section, an overview of the related work will be given. First off, the more relevant results of a literature survey will be presented, followed by a description on the state of the art currently in use in the space industry.

3.1 Academic Research

Approaches on key management in DTN can be roughly organized into one of two categories: decentralized approaches, similar to peer-to-peer architectures, and centralized approaches, comparable to the terrestrial Public Key Infrastructures. Since we want a centralized solution, the first group of approaches will not be further investigated here.

There are several approaches which utilize the central instance to generate and distribute shared secrets:

Besien [8] suggests sorting nodes into groups with one "Group Authority" per group, distributing secret information to each group member. Using this information, it is possible for nodes to securely communicate with other group members.

Ding et al. [9] and Liu et al. [10] introduce a "Key Generation Center" and "Key Distribution Center" respectively. These are contacted by nodes to retrieve key information. In addition to the central authority, [10] introduces "Distributed Key Distribution Centers" to keep the nodes up-to-date during system runtime.

In [11], a “police officer” is used to generate a “digital ID card”. This digital ID card will be signed by the node itself to be safe from counterfeiting.

Common to the approaches above is the central instance which generates and distributes shared secrets or even full keys. A potential risk is that this can lead to attackers having knowledge of all secret information if the central instance is compromised.

Shikfa, Önen, and Molva [12] investigate a more local approach, focusing on the prevention of Sybil attacks. Here, an “Identity Manager” initially uses a challenge-response exchange to verify the ownership of a public key received by a node. After the initial verification, the node uses a secure neighbourhood discovery algorithm which also includes a local key agreement protocol relying on signature chains. Advantageous with this approach is that no central instance knows all secret information, resulting in less leaked information in case of compromise.

Aside from concrete approaches for key management, Bhutta, Cruickshank, and Nadeem [13] define four requirements for a “Key Management Architecture for DTN (*KMAD*)”. They recommend establishing a shared state without pre-loading keying material and minimizing the number of roundtrips. They further suggest employing multiple key-exchange algorithms and supporting various cipher suites to accommodate for asymmetric and symmetric algorithms.

Menesidou and Katos [14] propose the usage of “slack time”, defined as the time nodes wait between messages, for security related tasks. The tasks they suggest doing during this time include certificate renewal, CRL updates or the establishment of session keys.

Finally, there are some works that look only into key revocation:

Bhutta, Cruickshank, and Sun [15] propose a change to CRL structure to decrease its size by the number of messages + 400 bytes.

With *V’CER* Koisser et al. [16] propose the use of sparse merkle trees for efficient certificate validation. Using their technique, only ~3KB are needed to store the revocation data for one million nodes over the timespan of one year.

Both approaches offer an improved CRL handling that will be useful with rising node numbers. However, in this work we focus on a scheme to support near- to mid-term expected use cases, in which such large node numbers, scalability and associated concerns are not addressed.

3.2 State of the Art in the Space Industry

Currently, to the best of our knowledge, there is no standardised or publicly available implementation for key management in DTN. The “Space Missions Key Management Concept” [17], published in 2011 by the Consultative Committee for Space Data Systems (CCSDS), mentions DTN but does not specify a "key management architecture for these networks". They do, however, predict scalability to “become a major issue” which may make the introduction of a public key system necessary.

There are multiple RFC drafts on the topic of key management in DTN. F. Templin and S. C. Burleigh [18] define nine core requirements and four design criteria in an expired RFC draft. Comparing these to the previously mentioned KMAD, it appears KMAD is much more general and an architecture originating from [18] might just be one of the different key exchange algorithms mentioned in the third requirement in KMAD.

Aligning closely with these requirements, Burleigh et al. [19] specify an architecture for “Delay-Tolerant Key Administration” (*DTKA*). This draft describes a distributed *Key Authority* consisting of multiple *Key Agents*. These receive updates from participating nodes and periodically reach consensus about recent updates to send out a “Bulletin” to all nodes containing those updates.

Another expired RFC draft, [20], describes a “Public Key Distribution Network” (*PKDN*) for DTN. This PKDN is an overlay network over DTN holding a current CRL. Any certificate must pass through this network to be validated. Receivers will only accept certificates sent through the network. The PKDN remembers the recipient of the bundle and informs it of updates or revocations of the certificate.

An extension for the Automated Certificate Management Environment (ACME) protocol is specified in [21], a currently active internet draft. This extension would allow an ACME server to validate a BP node’s Endpoint ID using a challenge-response pattern via an ACME client.

All these drafts bring some useful concepts. DTKA and PKDN make the central authority redundant by introducing multiple *Key Agents* and PKDN-Routers, respectively. PKDN especially mentions an interesting feature where nodes

can subscribe to key updates. Problematic with both is their requirement of multiple spacecraft acting as central nodes, possibly leading to very high initial costs and delays in setting these up. The ACME extension automates the bootstrapping phase by automatically validating the endpoint ID and issuing certificates to the nodes. Even though this approach is highly interactive, it is practical in automatically confirming a nodes identity and binding it to a certificate but does not solve the later key distribution/handshake problem of communicating nodes.

4. Design

Our approach for the Bundle Protocol Key Distribution (BPKD) is designed with the near- to mid-term deployments in mind, meaning a comparatively small number of nodes (<100). The following section gives a short overview of all parts of the system. The section after describes the internal workings of each node and how the key distribution application interacts with the Bundle Protocol Agent (BPA). Afterwards comes a section describing the interaction between the nodes and how the key distribution works. The final two sections explain the protocol messages used in the system and some points related to security .

4.1 Overview

The design is centralized (REQ3.4), meaning there is a central authority managing the keys and distributing them to the nodes. This central node is called the *Key Authority*. It collects updates, i.e. registrations, revocations and key updates, from the *Clients* and human operators and periodically sends out a bundle to all *Clients* containing this information and additional metadata, called a *Snapshot*. All *Client* nodes and the *Key Authority* belong to one *Application Domain*. Possible examples for domains would be space agencies, private companies, governments or similar. *Clients* register with their domain's *Key Authority* and receive public keys from it. To allow for secure communication between domains, a *Client* can take the role of a *Subscriber* and subscribe to key updates from a *Key Authority* outside its domain. This relationship is shown in Figure 1.

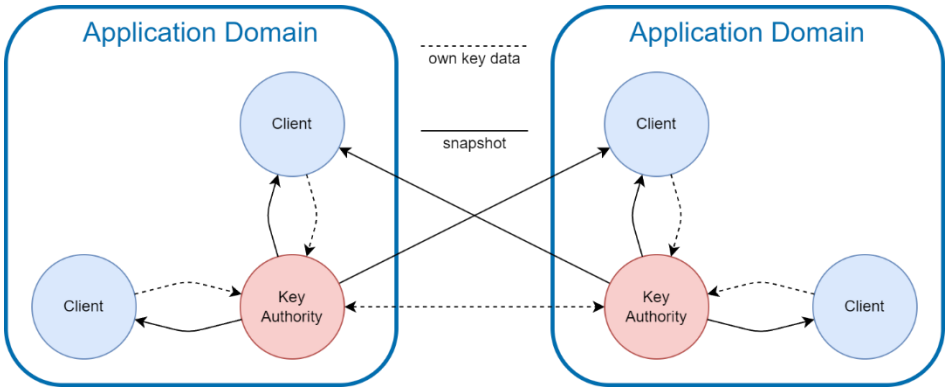

Figure 1: Overview of different roles in BPKD.

Displayed are two *Application Domains* with their respective *Key Authority* and two *Clients* each. The arrows indicate the direction and type of the key exchange: dashed arrows indicate the node's own *Key Data*. This *Key Data* must include the public key, the node's endpoint ID, and two timestamps: *not_before* and *not_after*, denoting the key's lifetime. Solid arrows indicate a collection of *Key Data*, i.e. a *Snapshot* containing the *Key Data* of every *Client* registered with the *Key Authority*.

Each *Client* sends its own *Key Data* to and receives *Snapshots* from the *Key Authority* within their domain. Additionally, the two upper *Clients* receive *Snapshots* from the other *Application Domain's Key Authorities*. Finally, the *Key Authorities* exchange their *Key Data* between each other, for reasons laid out later in 4.4.6.

The keys exchanged in this system are the public keys (REQ1.2) of an asymmetric key pair generated by the nodes themselves. However, to be compatible with the default security contexts as required by REQ1.1, the BPA needs to be provided with symmetric keys. To achieve this, we introduce a "Two-Layer Architecture" in which key management and key usage are separated. The key management layer uses and distributes public keys, the BP layer then uses the public keys to derive symmetric keys for the BPSec handler. This separation will be explained in more detail in 4.3.

4.2 Components

The following subsections describe the two components of this system, namely the *Key Authority* and the *Client*. An overview of the role they take in the system, their tasks and the use-cases they offer is given.

4.2.1 Key Authority

The *Key Authority* takes the central role within one domain, distributing the keys of all *Clients* in this domain. As opposed to some approaches mentioned in 3 ([8]-[11]), the *Key Authority* will not generate or distribute secret material but instead focus solely on the distribution of *Key Data*.

The *Key Authority* has two tasks: listening for updates from *Clients* or operators, and regularly sending out *Snapshots* to *Clients* and *Subscribers*. Updates include newly registered *Clients* (see 4.4.2), roll-over messages from *Clients* to update their public key (see 4.4.3), or manually triggered revocations (see 4.4.5). These updates are applied to the *Key Authority's* local key store. On a regular interval, the current state of the key store is sent to all *Clients* in the form of a *Snapshot*.

Compared to DTKA [19], where the key authority consists of multiple key agents connected with a sub-second one-way-light-time link, we propose a singular terrestrial *Key Authority* per domain. The reasons for this are manifold. First, it is faster and cheaper to deploy than an off-world distributed *Key Authority*, since we do not need to commission new or update existing spacecraft. Second, it is likely easier to harden a server deployed and under terrestrial facility control compared to implementing similar controls on an off-world system or spacecraft. Third, it is easier to achieve a high reliability and availability by being able to replicate the application and database on multiple servers. Finally, a terrestrial *Key Authority* can use the existing network of ground stations to achieve a very wide coverage with essentially only one node.

4.2.2 Client

The *Clients* regularly receive key updates in the form of *Snapshots* from the *Key Authority* and use the received *Key Data* to derive shared symmetric keys. For this, the *Client* regularly needs to check which public keys have turned valid to then derive the symmetric keys, as described in 4.3.

To facilitate inter-authority communication, *Clients* can subscribe to a *Key Authority* other than their own to receive *Snapshots* from it. In this case, *Clients* will be called *Subscribers*.

4.3 Two-Layer Architecture

As mentioned earlier, to stay compatible with the default security contexts while also adhering to REQ1.2 (Keys Should Be Asymmetric), we utilize a two-layer architecture: the actual key exchange happens on the asymmetric layer, while the encryption takes place on the symmetric layer. The interaction between the asymmetric and symmetric layers is shown in Figure 2.

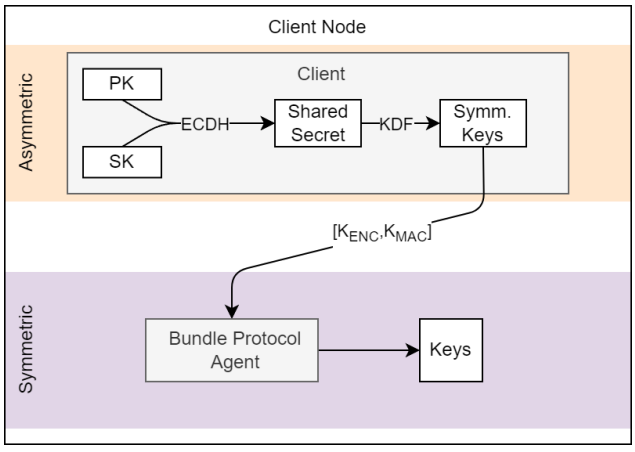

Figure 2: Internal client node interactions.

The upper, orange coloured layer is the asymmetric layer while the lower purple layer is the symmetric layer. To provide the BPA with symmetric keys, the *Client* application running on the BP node uses the Elliptic-curve Diffie-Hellman (ECDH) protocol to derive a shared secret from its own secret and another node's public key. The shared secret is then used as input for a Key Derivation Function (KDF) which produces symmetric AES keys. These keys are then sent to the BPA, to store them for later use with BPSec. This means there are two separate keystores on each node: a keystore on the asymmetric layer and a keystore on the symmetric layer. Only the latter stores the keys to be used for actual BPSec operations.

Currently, the symmetric keys are used for BPSec operations directly. However, this introduces security implications, the main one being the long lifetime of the derived symmetric keys. Since the symmetric keys are derived directly from the asymmetric keys, they share the same, possibly very long, lifetime. Since this is not ideal from a security perspective, it is likely necessary to take a deeper look at how the derived keys are used. Possible options would be the usage of the derived keys as master keys for key encryption or to derive further symmetric keys with a shorter lifetime. Alternatively, the usage of a ratcheting scheme similar to the signal protocol [28] or the messaging layer security protocol [29] could be approaches to achieve Perfect Forward Secrecy (PFS) and Post-compromise Security (PCS). This is identified as promising future work.

This two-layer architecture can in principle also support exchange of algorithms, as required in REQ3.6 i.e. key agreement and derivation algorithms should be adaptable to fit the needs and to stay future proof, as long as they output the required symmetric AES keys. The key management in itself and the BPSec functionality do not need to be changed, only the application's part handling the derivation of the symmetric keys needs adaption.

As described, the design is compatible with the default security contexts. However, it is likely for more BPSec security contexts to be specified in the future, especially security contexts supporting asymmetric cryptography. One step in that direction is taken with the draft for the COSE context [22]. This security context enables the use of COSE [23] algorithms in BPSec. The implementation of such a security context would enhance and complement BPKD significantly as instead of having two separate keystores, only the BPSec keystore would be needed. The BPKD architecture would not need to derive a set of symmetric keys to send to the BPA. Instead, the public keys received from the *Key Authority* would be sent to the BPA directly. It would also allow for improved low-level key management and proper digital signatures. The former would alleviate the problem of the long lifetimes of derived keys, since a new symmetric key would be derived more frequently. The latter would simplify the key distribution further since the BPA would not need to send separate bundles to each *Client/Subscriber* with a unique message authentication code for each. Instead, it would suffice to apply a BIB with a digital signature to the bundle. Every receiver of the bundle in possession of the *Key Authority's* public key would be able to verify the bundle. This would also allow for multicast transmissions of *Snapshots*.

4.4 Interaction

This section describes how the system components interact with each other, including the initial setup of a new node, the key distribution, revocation and rollover functionalities of *Clients*, as well as the inter-domain key exchange. First, however, a description of what is required for the successful deployment of a node is provided.

4.4.1 Prerequisites

Of key importance for on-board functionality is the ability for each node to generate new key-pairs. This comes with the requirement for a true random number generator. Alternatively, operators could upload secret material using a secure channel. This is likely less secure however, as it implies a wider attack surface. Furthermore, nodes need to implement a (possibly hardware-accelerated) key derivation function.

There also needs to be a secure out-of-band channel to the node. Before launch, direct physical access with portable media could be used. After launch, use of the Space Data Link Security (SDLS) [24] protocol could be an option.

Finally, the *Client* application needs to be loaded onto the node and initial trust needs to be established. This establishment of initial trust is described in the next subsection.

4.4.2 Initial Setup

To add new nodes to the system, as required by REQ3.2, there needs to be an initial setup to establish trust. First, the *Client* application needs to be loaded onto the spacecraft and configured properly. Next, initial trust needs to be

established. For that, *Key Data* is exchanged between the *Key Authority* and the *Client* using a secure channel as described in chapter 4.4.1. It is assumed that such a channel exists.

Once this initial trust is established, the nodes can verify the authenticity of each other's bundles by validating the MAC or signature, depending on the algorithms used.

Optionally at this step, the *Client* can be provided with a current *Snapshot*, so it does not need to wait for the *Key Authority*'s periodic *Snapshot*.

A similar procedure needs to be executed in case a node is compromised or after it could not update its *Key Data* before expiration. After control has been regained, the node needs to be re-added to the system using this process.

As previously mentioned, an option to achieve this with a degree of automation could be to incorporate the ACME protocol for DTN EID validation [21]. Although this protocol is very interactive, it allows the automatic validation of EIDs and the automated creation of certificates for BP nodes.

4.4.3 Key Rollover

Since keys must have a lifetime (REQ1.3), *Clients* must be able to update their key pair from time to time (REQ3.3). For this, there is a rollover mechanism in place. Early enough before their current key pair expires, *Clients* generate a new key pair and send the new *Key Data* to the *Key Authority*. The *not_before* date should be chosen in a way for the *Key Data* to be received by every node in the network by the time it turns valid. This assumes some degree of time synchronization within the network.

REQ1.4 further requires the system to be able to handle multiple keys per node. To achieve this, the *Key Authority* (and therefore all its *Clients*) store other node's keys not only under the node's ID, but also with the key's *not_before* date. This makes the keys easily distinguishable by their lifetime. Furthermore, expired keys are not deleted immediately, but after a configurable grace period. This enables receiving nodes to compare the creation timestamp of a bundle to the lifetime of a key and to use a deactivated key to verify cryptographic operations. Figure 3 shows this grace period and possible different key states which are based on CCSDS SDLS Extended Procedures [31]. The duration of the grace period may be subject to security risk assessment considerations accounting for possible misuse scenarios.

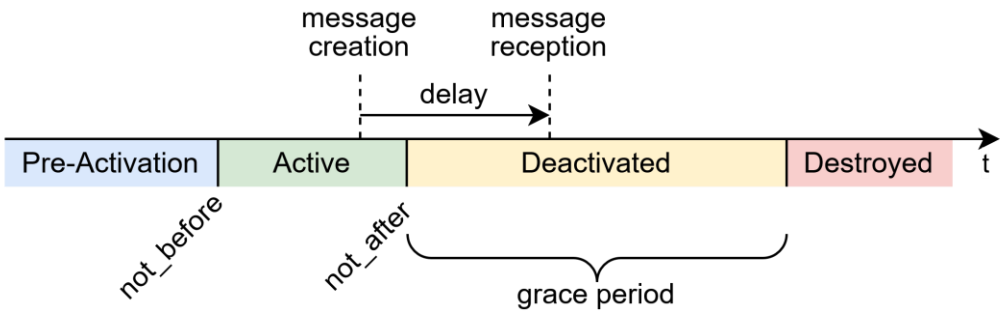

Figure 3: Key states.

Before the key's *not_before*-date, it is in the "Pre-Activation" state. Once the *not_before*-date is reached, it enters the "Active" state and may be used to apply and verify cryptographic operations. After the *not_after*-date passes, the key assumes the "Deactivated" state and the grace period starts. During this period, the key may only be used to verify cryptographic operations. After the grace period, the key is destroyed. If the bundle was created while the key was still active, the deactivated key may still be used. The details of this, however, are left to the BP implementers and especially the BPSec part thereof.

4.4.4 Key Distribution

The *Key Authority* collects all registrations, rollover and revocations and applies the changes to its local keystore. On a regular interval, the *Key Authority* sends a *Snapshot* of its keystore to all its *Clients* and *Subscribers*. This fulfils REQ2.1, since all participating nodes will receive a *Snapshot*. To use multicast for this message (REQ2.5), the message needs to be the same for every receiver. This is achieved by sending the same *Snapshot* to all *Clients*. Further, each *Subscriber* receives the same message (for details see 4.4.6), meaning multicast can also be utilized to send *Snapshots* to *Subscribers*.

Since *Clients* choose the lifetime of new keys in a way making sure keys are propagated through the network before they turn valid, REQ2.3 is fulfilled as well. This *Snapshot* mechanism leads to all *Clients* having every other *Client's* public key before it is valid. This means *Clients* do not need to perform a handshake before they can send and receive bundles, fulfilling REQ2.4. All of this happens automatically in a regular interval, fulfilling REQ2.2.

4.4.5 Key Revocation

As specified by REQ3.1, the system must allow for key revocations to limit key misuse. For this, the *Key Authority* offers an interface to the operator, who needs to manually revoke a key. Once this has been triggered, the *Key Authority* immediately sends a revocation message to all *Clients*. Additionally, the *Key Authority* applies the revocation to its keystore, such that the change is also reflected in the next *Snapshot*.

4.4.6 Inter-Domain Key Exchange

To accommodate for collaboration between different space agencies and other entities which may have their own *Application Domain*, the system must support key exchange between multiple *Key Authorities*, as required by REQ3.5. To achieve this, *Clients* can subscribe to updates from a *Key Authority* outside their own *Application Domain*. In this context, a *Client* who subscribed to a *Key Authority* will be called a "*Subscriber*". Once a *Key Authority* receives such a subscription, it sends a filtered version of its current *Snapshot* to the *Subscriber*. This filtered version only contains *Key Data* of *Clients* that are supposed to be publicly known. The *Key Authority* remembers the *Subscriber* as a target for future *Snapshot* and revocation messages.

To guarantee security, a *Subscriber* needs to confirm the authenticity of the bundles received from the other domain's *Key Authority*. For this to work, the *Subscriber* needs access to the *Key Authority*'s *Key Data*, requiring every *Key Authority* to know every other *Key Authority*'s *Key Data* to be able to distribute it to its *Clients*. This also means there must be a way to securely facilitate cooperation between *Key Authorities*. This is, however, out of the scope for this work, we assume manual configuration between *Key Authorities*. One step in that direction is taken by the *Security Working Group* of the *Systems Engineering Area* (SEA-SEC) of the CCSDS with their work on an *Intergovernmental Certification Authority* [25].

Each *Key Authority* may decide for itself which *Key Data* it sends to which *Subscriber*. It is entirely possible that there are some *Key Data* which are only allowed to be distributed within an *Application Domain*. Similarly, nodes are only allowed to subscribe to *Key Authorities* allowed by their own *Key Authority*.

In the same way it is possible for a *Client* to subscribe to a *Key Authority* outside its domain, it must be possible for a *Subscriber* to unsubscribe from the *Key Authority*. If the *Subscriber* would stay subscribed, even if the *Snapshots* are not needed anymore, the bundles sent to the *Subscriber* would be unnecessary overhead. Another feature to reduce unnecessary overhead on the network and the *Key Authority* is the need to resubscribe periodically to keep receiving *Snapshots*. In the same sense as before, "dead" *Subscribers* lead to unnecessary traffic when *Snapshots* are sent to them, even though they are offline.

Both mechanisms can be disregarded when using multicast: since the multicast bundles will be sent through the network anyway and have less overhead than multiple unicast bundles, *Clients* can simply forward multicast bundles without delivering them locally.

4.5 Message Format

This section focuses on the different BPKD protocol messages. The next section specifies the encodings for the different BPKD message types and Section 4.5.2 discusses the difference between *Snapshots* and *Bulletins* and the reason for using *Snapshots* over *Bulletins*.

4.5.1 Encoding

All BPKD messages are encoded in a CBOR array with the first entry being an unsigned integer, specifying the type of the message. This value is followed by a sub-array with message-specific data. Both values must be present, even if there is no message-specific data. In this case, the sub-array is empty. Table 1 shows the different messages, their type codes and a short description of each message.

Table 1: BPKD message types.

Code	Name	Description
1	UPDATE	Informs KA about a key update.
2	SNAPSHOT	Contains the KA's current keystore.
3	REQUEST-SNAPSHOT	Request a *Snapshot* if waiting is not possible.
5	REVOKE	One or more keys to be revoked.
10	SUBSCRIBE	Signals interest in an external KA's data.
11	UNSUBSCRIBE	Informs KA that no further *Snapshots* are required.

4.5.2 Snapshot vs Bulletin

Since *Snapshots* contain every *Client's Key Data*, they linearly grow with each new *Client*. Especially when using large keys in a scenario with a lot of nodes, this may lead to large *Snapshot* sizes.

An alternative message format to the *Snapshot* described in [19] is the "Bulletin". Instead of containing the current *Key Data* for every *Client* in each message, the *Bulletin* only contains data which has changed since the last *Bulletin*. On receipt of a *Bulletin*, a *Client* will update its local keystore with the new information. The advantage of using *Bulletins* is the reduced message size, especially with larger amounts of nodes: since not every *Client's Key Data* is contained in the messages, the messages are much smaller. The actual factor depends on the number of updates happening in the time frame between two *Bulletins*.

Bulletins, however, have some unique drawbacks. Firstly, state management becomes more complex for the *Key Authority* and the *Client*. The *Key Authority* must track which updates were sent in each *Bulletin*, and *Clients* must track which *Bulletins* they have received to detect any missed updates. This leads to the second issue: if a *Bulletin* is lost, the *Client* will only notice the missing update when the next *Bulletin* arrives. At that point, the *Client* must request the missed *Bulletin*, resulting in additional round-trip time and increased overhead.

To avoid these drawbacks and since we focus on the near- and mid-term, we will use the more basic *Snapshot*.

4.6 Security

To maintain a high level of security, several measures need to be considered. First, BPKD messages must be protected using BPSec's Bundle Integrity (BIB) and Confidentiality (BCB) Blocks. This ensures data-in-transit is safeguarded against tampering and unauthorized access, while also providing implicit authenticity. When using the BCB-AES-GCM security context with a key size of at least 256 bits, confidentiality can be maintained even in a post-quantum setting.

Second, it is important that only operators or trusted third parties can register nodes as *Clients*. This prevents unknown and potentially malicious nodes from gaining information about the *Application Domain*. Another step in maintaining confidentiality for secret nodes is the *Key Authority's* ability to limit what *Key Data* is sent to *Subscribers*. Similarly, the *Key Authorities* must be able to limit which other *Key Authorities* its *Clients* can subscribe to. This prevents malicious *Key Authorities* from injecting deliberately false *Key Data* into *Clients*.

For node-internal security it is important that the data transmission between application agent and BPKD application is authenticated and encrypted. This needs to be done to protect the data from other, potentially malicious, applications running on the node. It is doubly important since over this channel, not only the *Key Data* is exchanged, but also the derived symmetric keys, possibly giving an attacker full access to the encrypted bundles. Additionally, data-at-rest needs to be protected by using a secure keystore/security module. This is especially important for the secret keys generated by BPKD.

Finally, to remain secure also in the future, it is important to keep the cryptographic algorithms exchangeable. This paves way for quantum-resistant algorithms, once they are required. The NSA, for example, already announced the *Commercial National Security Algorithm Suite* 2.0 [27], requiring exclusive use of post-quantum algorithms by 2033 at the latest, with some specific areas already requiring them by 2030.

Two properties considered out-of-scope for this concept are PFS and PCS, which protect communication even in the event of compromise. Both require a more detailed low-level key management concept, i.e. management of the symmetric keys after they have been derived.

5. Evaluation

To evaluate this approach, a case study has been utilized. The following section looks into the setup of the case study. Section 5.2 describes the two scenarios in which the prototype has been deployed. The section afterwards describes different test cases used for the evaluation and the final section gives insight into the results.

5.1 Setup

To evaluate the design and prototype, BPKD was deployed on nodes running ESA's BP implementation. Each node is modelled by a docker container, with all connections between two nodes represented by individual docker networks. To emulate the challenges in delay tolerant networks, the properties of these docker networks can be adapted to fit the requirements, e.g., periodic link loss and transfer delays according to provided contact plans.

5.2 Scenarios

The scenarios in which BPKD was evaluated are the *Earth Observation* and a *Lunar Communications* DTN scenarios from [32]. Both were developed internally by ESA with the long-term goal of having a common set of scenarios to evaluate BP technology in. The topologies and contact plans were all derived from past and planned future missions from different agencies.

The first scenario, *Earth Observation*, is a rather simple scenario. It consists of a mission and payload control centre, two ground stations, and an earth observation satellite. The satellite has intermittent connectivity to the ground stations, which are both connected to the mission and payload control centre. The *Key Authority* role is taken on by the *Mission Control Center (MCC)*, with the other nodes being *Clients*. A diagram of the *Earth Observation* scenario can be seen in Figure 4.

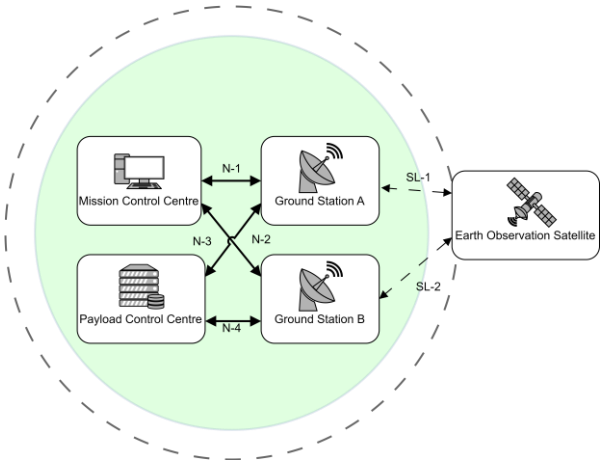

Figure 4: The Earth Observation scenario.

The more complex *Lunar Communications* scenario can be seen in Figure 5. It contains five terrestrial control centres, two ground stations, the *Lunar Gateway* with an onboard user, two relay satellites, and two lunar assets. Intermittent connectivity characterizes all links involving extra-terrestrial nodes, specifically those located beyond the ground stations in the network topology. The colours in the diagram show which control centre is responsible for which node: the base control centre is responsible for the lunar base, the user control centre for the user, etc.

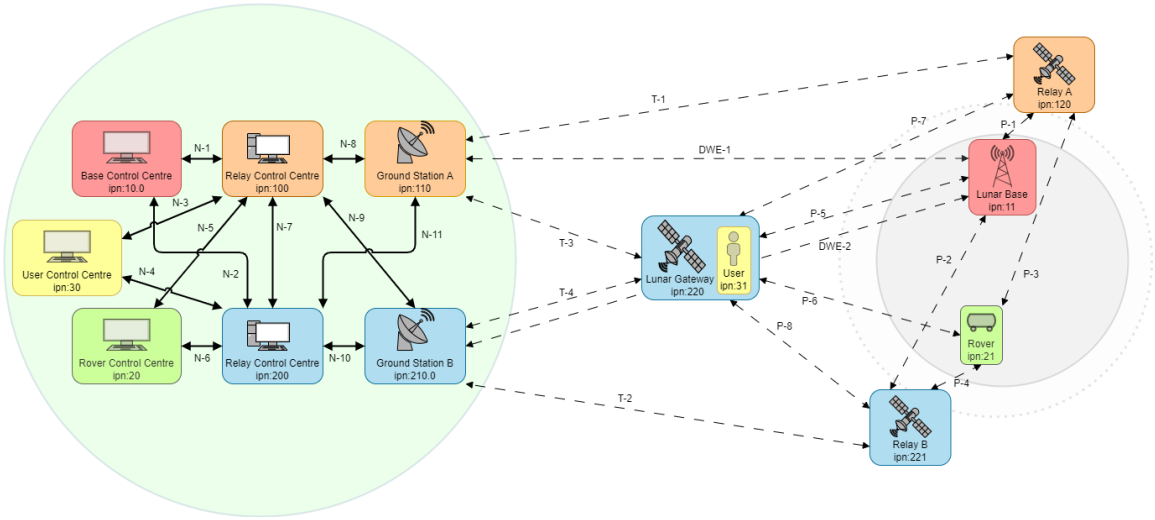

Figure 5: The Lunar Communications scenario.

This scenario contains two *Key Authorities* that are deployed on each *Relay Control Centre (RCC)*. To model inter-authority communication, some *Clients* also subscribe to the other *Key Authority*: the rover control centre is subscribed to *RCC B* (ipn:200), while *Relay B* and *Lunar Gateway* are subscribed to *RCC A* (ipn:100). This is due to the hypothetical limitation that the rover does not have enough computational power to perform complex cryptographic calculations itself, which is why the *Lunar Gateway* and *Relay B*, as the first forwarding nodes, may be required to perform these calculations. For this to work, both senders (*Lunar Gateway* and *Relay B*) and the receiver (*Rover Control Centre*) require each other's *Key Data*, resulting in the need to be subscribed to each other's *Key Authority*.

5.3 Test Cases

To evaluate the prototypes functionality, different test cases have been derived for events the system needs to be able to handle.

TC1 **Nodes Joining**
The first test case is the very basic event of a new node joining the system. For this, the nodes' out-of-band channel (in the case of the prototype, the HTTP interface) is used to perform the actions as described in 4.4.2. This test counts as successful when the *Client's* and *Key Authority's Key Data* has been exchanged, allowing the *Client* to partake in future key exchanges.

TC2 **Key Rollover**
After a node has joined, it can perform a rollover to renew its public key pair. The *Client* performing the rollover follows the procedure outlined in 4.4.3 to send its new *Key Data* to the *Key Authority*. Once the *Key Authority* has received the new *Key Data*, this test case can be seen as successful.

TC3 **Key Distribution**
Regularly, the nodes' *Key Data* needs to be distributed to all other *Clients*, a process modelled by this test case. It is also rather simple and follows the procedure described in 4.4.4. When all *Clients* receive a *Snapshot*, this test case has succeeded.

TC4 **Revocations**
In case a *Client* is compromised, the *Client's Key Data* needs to be revoked to prevent abuse of the key. This test case is used to show that key revocation works and is considered successful when the *Client's Key Data* has been removed from every *Client* and the *Key Authority*.

TC5 **Re-establishing Trust**
If a *Client* fails to update its key pair in time and its most recent *Key Data* expires, or if a *Clients Key Data* has been revoked, it needs to be re-added to the *Application Domain* and trust to the *Key Authority* needs to be re-established. TC5 test case verifies this process. It is considered successful when the *Client* is part of the *Application Domain* again.

TC6 **Inter-Authority Communication**
The final and most complicated test case is for "Inter Authority Communication". This test case proves that the system not only works with one, but also with multiple *Application Domains*. It tests the functionality of *Clients* to subscribe to external *Key Authorities*. Success is measured by *Clients* being able to use BPSec with other *Clients* outside their *Application Domain* and by *Key Authorities* keeping *Subscribers* up-to-date w.r.t current *Key Data*.

5.4 Results

After running our network testbed and evaluating different scenarios and test cases, the following conclusions can be drawn:

It is easily possible to add new *Clients* to the system using the described HTTP interface (TC1). Once added, *Clients* autonomously update their key pairs and perform regular rollover operations within the configured interval to have a current key pair available and to keep the *Key Authority*, and by extension all other *Clients* within the *Application Domain*, up to date (TC2). An important role in keeping *Clients* up to date is played by the *Key Authority* regularly and autonomously sending *Snapshot* bundles to all *Clients*. In the end, every *Client* can use the received *Key Data* to apply BPSec blocks to the bundles sent to other *Clients* in the *Domain*, successfully fulfilling TC3.

TC4, the revocation of keys, works successfully too. It is possible to revoke keys, effectively removing nodes from the *Application Domain* and rendering them unable to apply or verify BPSec blocks. Currently, each key needs to be revoked manually using the *Key Authority's* respective HTTP endpoint. For ease of use it might make sense to introduce functionality to revoke all keys for one node at once. Further, instead of immediately deleting the keys as is done currently, they could be put into a suspended state until compromise has been confirmed.

The fifth test case, re-establishment of trust between a node and the *Key Authority*, succeeds too. Using the node's and *Key Authority's* HTTP interface, the *Key Data* can be exchanged manually to register the node as a *Client* again. It should be said, however, that this test case is especially simple when compared to the real-world operation, which requires the use of a secure out-of-band connection, such as the SDLS protocol, to the node to re-establish trust, complicating the whole process.

Even the final and most complicated test case, *Inter Authority Communication*, also works well. Once all nodes are set up correctly, it is possible to send BPSec secured messages to nodes in another *Application Domain*. To test this, the scenario described in 5.2 has been set up. When the rover sends a bundle to its control centre, it does not apply BPSec operations to the bundle. Instead, the first forwarding node (*Relay A*, *Relay B* or the *Lunar Gateway*) performs this operation on the received bundle and forwards the bundle to the *Rover Control Centre*. If the first forwarding node is either *Relay B* or the *Lunar Gateway* this test case can be tested, as these nodes are not in the same *Application Domain* as the *Rover Control Centre* – hence, inter-authority communication is required for the forwarding nodes to have the required key for the rover control centre. All of this works, and the *Rover Control Centre* is able to verify the security operations applied by a node outside its *Application Domain*, proving that TC6 is successful too.

In summary, the prototype works well and successfully handles all test cases. These range from simple scenarios like the registration of new nodes to complex scenarios like the key exchange between agencies. Although the testbed environment used to test the prototype in is much simpler than the space environment an actual implementation would have to work in, this demonstrates that the proposed approach can fulfil its task of key distribution in DTNs while coping with the unique challenges in DTN.

6. Conclusion & Outlook

This paper presents an experimental architecture for key distribution in DTNs called BPKD. It addresses the challenges inherent to DTNs, such as delays, disruptions, and more, while maintaining compatibility with the default security contexts and remaining agnostic to the cryptographic algorithms and distributed keys. It is well-suited for the near- to mid-term by using a terrestrial *Key Authority* yet remains scalable should the need for an off-world, distributed *Key Authority* arise. Finally, it supports inter-domain key exchange and delay-tolerant key revocations.

To achieve this, requirements for a key management architecture in DTNs have been stated. The requirements consider the challenges in DTNs and serve as a base for the design. The design consists of the central *Key Authority*, collecting updates and distributing keys on a regular interval, and its *Clients*, which register with the *Key Authority* and regularly receive a *Snapshot* of all keys known to the *Key Authority*. Both parties use a two-layered architecture to stay compatible with the default security contexts while being agnostic to the cryptographic algorithms.

As shown in the case study, the proposed approach works and handles delays and disruptions well, especially with long key lifetimes and timely rollovers. Inter-authority communication is also facilitated by BPKD while retaining security by limiting what outsiders can learn about the network structure and by limiting which external *Key Authorities Clients* trust.

An important point that was not elaborated more closely in this work is the low-level key management, i.e. how the derived symmetric keys are to be used. Currently, they are used directly with BPSec, leading to long key lifetimes which decreases security. This also means there is currently no PFS or PCS possible. While there have been some ideas on how to handle this there is no concrete design or implementation yet.

New security contexts, especially ones supporting asymmetric cryptography, will further improve the functionality of BPKD. The upcoming BPSec COSE context, offering digital signatures and enveloped messages, would simplify the current architecture and enable confidentiality, integrity, and authenticity in multicast messages. Apart from that, new security contexts can also offer post-quantum secure signature and key-agreement schemes once post-quantum resistance is required. Repeat testing of scenarios with evolved contexts and standardised PQC implementations is envisaged as future work.

In conclusion, BPKD is considered a step in the right direction for key distribution in DTNs with the near- to mid-term in mind. The flexibility, especially in the message format and cryptographic algorithms, aids future proofing by being able to adapt to rising node numbers and stronger security requirements. While there are areas for improvement, such as conducting formal security proof assessments, improving representativeness of the testbed protocols stack (e.g. for out-of-band channels), and defining low-level key management concepts and PFS/PCS, it offers a solid foundation for further development and research of key management in DTNs.

References

[1] S. Burleigh, K. Fall, and E. Birrane, “Bundle protocol version 7,” Internet Engineering Task Force, 2022. doi: 10.17487/RFC9171.

[2] “The future lunar communications architecture.” IOAG, 2022. Available: https://www.ioag.org/Public%20Documents/Lunar%20communications%20architecture%20study%20report%20FINAL%20v1.3.pdf

[3] “The future mars communications architecture.” IOAG, 2022. Available: https://www.ioag.org/Public%20Documents/MBC%20architecture%20report%20final%20version%20PDF.pdf

[4] “LunaNet Interoperability Specification Draft Version 5,” 2023. Available: https://www.nasa.gov/wp-content/uploads/2023/09/lunanet-interoperability-specification-v5-draft.pdf

[5] E. Birrane and K. McKeever, “Bundle protocol security (BPSec),” Internet Engineering Task Force, 2022. doi: 10.17487/RFC9172.

[6] R. C. Durst, G. J. Miller, and E. J. Travis, “TCP extensions for space communications,” International conference on mobile computing and networking (MobiCom). ACM, 1996. doi: 10.1145/236387.236398.

[7] G. Koukis, K. Safouri, and V. Tsaoussidis, “All about delay-tolerant networking (DTN) contributions to future internet,” Future Internet, vol. 16, no. 4, 2024, doi: 10.3390/fi16040129.

[8] W. L. V. Besien, “Dynamic, non-interactive key management for the bundle protocol,” ACM workshop on challenged networks, ACM, 2010. doi: 10.1145/1859934.1859951.

[9] Y. Ding, X. Zhou, Z. Cheng, and W. Zeng, “Efficient authentication and key agreement protocol with anonymity for delay tolerant networks,” Wireless Personal Communications, vol. 70, 2013, doi: 10.1007/s11277-012-0760-x.

[10] Y. Liu, A. Zhang, J. Li, and J. Wu, “An anonymous distributed key management system based on CL-PKC for space information network,” IEEE international conference on communications (ICC), IEEE, 2016. doi: 10.1109/ICC.2016.7510841.

[11] Y. Xie and G. Wang, “Practical distributed secret key generation for delay tolerant networks,” Concurrency and Computation: Practice and Experience, vol. 25, 2013, doi: 10.1002/cpe.2885.

[12] A. Shikfa, M. Önen, and R. Molva, “Local key management in opportunistic networks,” International Journal of Communication Networks and Distributed Systems, vol. 9, 2012, doi: 10.1504/IJCNDS.2012.047898.

[13] M. N. M. Bhutta, H. Cruickshank, and A. Nadeem, “A framework for key management architecture for DTN (KMAD): Requirements and design,” International conference on advances in the emerging computing technologies (AECT), IEEE, 2020. doi: 10.1109/AECT47998.2020.9194164.

[14] S. A. Menesidou and V. Katos, “Opportunistic key management in delay tolerant networks,” International Journal of Information and Computer Security, vol. 9, 2017, doi: 10.1504/IJICS.2017.085136.

[15] M. N. M. Bhutta, H. Cruickshank, and Z. Sun, "Public-key infrastructure validation and revocation mechanism suitable for delay/disruption tolerant networks," IET Information Security, vol. 11, 2017, doi: 10.1049/iet-ifs.2015.0438.

[16] D. Koisser, P. Jauernig, G. Tsudik, and A.-R. Sadeghi, "V'CER: Efficient certificate validation in constrained networks," USENIX security symposium (USENIX security), 2022. Available: http://arxiv.org/abs/2205.01973

[17] CCSDS, Space missions key management concept. 2011. Available: https://public.ccsds.org/Pubs/350x6g1.pdf

[18] F. Templin and S. C. Burleigh, "DTN security key management - requirements and design." Internet Engineering Task Force, 2016. Available: https://datatracker.ietf.org/doc/draft-templin-dtn-dtnskmreq/00/

[19] S. C. Burleigh, D. Horres, K. Viswanathan, M. Benson, and F. Templin, "Architecture for delay-tolerant key administration." Internet Engineering Task Force, 2018. Available: https://datatracker.ietf.org/doc/draft-burleigh-dtnwg-dtka/02/

[20] K. Viswanathan and F. Templin, "Architecture for a delay-and-disruption tolerant public-key distribution network (PKDN)." Internet Engineering Task Force, 2015. Available: https://datatracker.ietf.org/doc/draft-viswanathan-dtnwg-pkdn/00/

[21] B. Sipos, "Automated certificate management environment (ACME) delay-tolerant networking (DTN) node ID validation extension." Internet Engineering Task Force, 2024. Available: https://datatracker.ietf.org/doc/draft-ietf-acme-dtnnodeid/14/

[22] B. Sipos, "DTN bundle protocol security (BPSec) COSE context." Internet Engineering Task Force, 2024. Available: https://datatracker.ietf.org/doc/draft-ietf-dtn-bpsec-cose/04/

[23] J. Schaad, "CBOR object signing and encryption (COSE)." Internet Engineering Task Force, 2017. doi: 10.17487/RFC8152.

[24] CCSDS, Space data link security protocol. 2022. Available: https://public.ccsds.org/Pubs/355x0b2.pdf

[25] CCSDS, Intergovernmental Certification Authority, 2024. Available: https://public.ccsds.org/Pubs/357x1o1.pdf

[26] E. Birrane, A. White, and S. Heiner, "Default Security Contexts for Bundle Protocol Security (BPSec)," Internet Engineering Task Force, 2022. Available: https://www.rfc-editor.org/info/rfc9173

[27] NSA, "Commercial National Security Algorithm Suite 2.0." NSA, 2024. Available: https://media.defense.gov/2022/Sep/07/2003071836/-1/-1/0/CSICNSA_2.0_FAQ.PDF

[28] J. Alwen, S. Coretti, and Y. Dodis, "The double ratchet: Security notions, proofs, and modularization for the signal protocol," in Advances in cryptology – EUROCRYPT 2019, Y. Ishai and V. Rijmen, Eds., Cham: Springer International Publishing, 2019.

[29] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon, 'The Messaging Layer Security (MLS) Protocol', Internet Engineering Task Force, 2023. doi: 10.17487/RFC9420.

[30] IPNSIG. Accessed: Mar. 06, 2025. Available: https://www.ipnsig.org

[31] CCSDS, *Space Data Link Security Protocol—Extended Procedures*. CCSDS, 2020. Available: https://public.ccsds.org/Pubs/355x1b1.pdf

[32] C. Malnati, F. Flentge, "Reference Scenarios for Evaluation of Disruption Tolerant Network Technologies", SpaceOps, 2025. (forthcoming)