IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 4, 26 pages.

https://doi.org/10.62056/a39qudhdj

Cryptography is Rocket Science

Analysis of BPSec

Benjamin Dowling[1], Britta Hale[2], Xisen Tian[2] and Bhagya Wimalasiri[3]

[1] King's College London, United Kingdom
[2] Naval Postgraduate School, United States
[3] University of Sheffield, United Kingdom

Abstract. Space networking has become an increasing area of development with the advent of commercial satellite networks such as those hosted by Starlink and Kuiper, and increased satellite and space presence by governments around the world. Yet, historically such network designs have not been made public, leading to limited formal cryptographic analysis of the security offered by them. One of the few public protocols used in space networking is the Bundle Protocol, which is secured by Bundle Protocol Security (BPSec), an Internet Engineering Task Force (IETF) standard. We undertake a first analysis of BPSec under its default security context, building a model of the secure channel security goals stated in the IETF standard, and note issues therein with message loss detection. We prove BPSec secure, and also provide a stronger construction, one that supports the Bundle Protocol's functionality goals while also ensuring destination awareness of missing message components.

Keywords: BPSec · Bundle Protocol · Space System Security · Satellite Security

1 Introduction

Originally developed for deep space communications, the Bundle Protocol Security (BPSec) [BM22] supports Delay Tolerant Networks (DTN)[Bir23] as an application layer secure channel protocol to facilitate a store-and-forward paradigm for sending messages between nodes. It was specifically designed with space system security in mind, a use case where security protocols used on the internet today, such as IPSec [FK11] and TLS [Res18], are sub-optimal due to latency, delays, and bandwidth constraints. BPSec was designed to be more suitable for distressed environments where delivery is not guaranteed, bandwidth is a premium, and roundtrip times are on the order of minutes or even hours. This has made it suitable for use outside of deep space networks with applications extending to the Internet of Things (IoT) [BBC17]. Such breadth of uses and interest has led BPSec to be standardized by the Internet Engineering Task Force (IETF). Yet, to date, there has been no formal cryptographic analysis of BPSec. This work provides a closer look at formalizing and strengthening space system channel security in a way that is appropriate under the intended use case constraints.

We give an example execution of BPSec in Figure 1. In this example, the satellites orbiting Earth act as bundle protocol agents (BPA) who create, forward and receive messages. The movement of these satellites and their availability to forward and receive messages are in a constant state of change which in turn creates a network infrastructure prone to high latency, connection disruptions and propagation delays. For instance in

E-mail: benjamin.dowling@kcl.ac.uk (Benjamin Dowling), britta.hale@nps.edu (Britta Hale), xisen.tian1@nps.edu (Xisen Tian), b.m.wimalasiri@sheffield.ac.uk (Bhagya Wimalasiri)

Received: 2024-10-07 Accepted: 2024-12-03

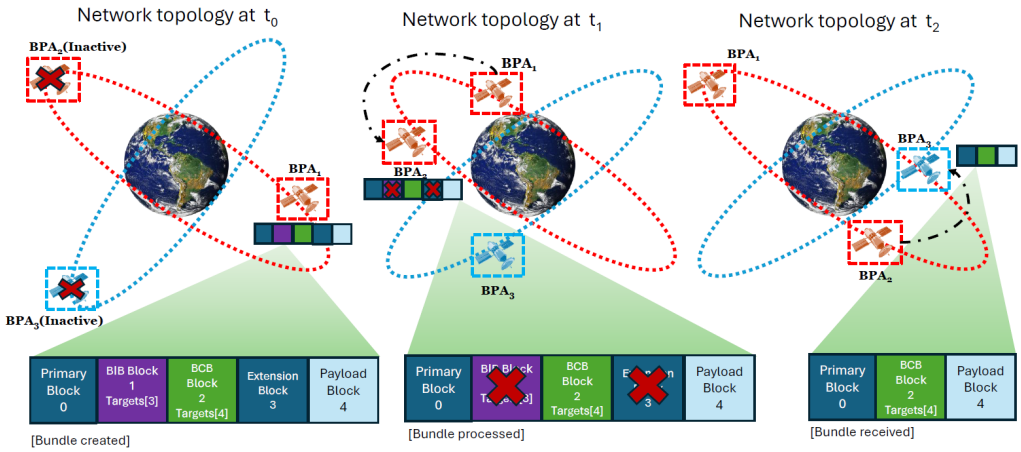

Figure 1: An execution of BPSec protocol. BPA_1 creates and forwards a bundle to BPA_2. BPA_2 processes blocks from the bundle, forwarding the modified bundle to BPA_3. This processing and discarding of bundle blocks by an intermediate BPA is legitimate behavior within a BPSec execution, provided that the corresponding intermediary has been authorized to process bundles by the relevant policies.

Figure 1, while BPA_1 creates the message at t_0, due to the unavailability of a viable path, it cannot be forwarded to the intended destination (BPA_3) immediately and thus stores the message until t_1. In this way, the unique network infrastructure BPSec operates within differs to the terrestrial internet. Within such DTN infrastructures, BPAs store messages for long periods and forward them once an appropriate link becomes available, until the message reaches its final destination.

Underlying BPSec is the Bundle Protocol, a DTN protocol analogous to other networking protocols such as TCP/IP. NASA has included Bundle Protocol version 7 in its three-phased approach for rolling out DTN-based communication links by 2030 [Man23b]. The Bundle Protocol requires security to be handled separately, a feature that BPSec provides while being inherently bound to the functionality requirements of the Bundle Protocol. Furthermore, customizable security contexts that define the ciphersuite and parameters used by BPSec participants allow for fine-tuning of the level of BPSec's security. Authentication and/or a combination of confidentiality and authentication are applied through use of Block Integrity Blocks (BIB) and Block Confidentiality Blocks (BCB). These blocks store the resultant tags of Message Authentication Code (MAC) and Authenticated Encryption with Associated Data (AEAD) calculations along with parameters used to generate them (i.e. nonces, SHA variant, wrapped key) [BWH22]. This concept of security reference blocks is intended to provide a great deal of flexibility to securing data blocks, where an intermediate node (e.g., a satellite) can add or strip security layers.

Notably, BPSec does not aim to provide key agreement nor does it establish freshness of a session in entity authentication, aiming to instead offer a secure channel protocol under the assumption of preestablished keys. It therefore lacks the common handshake component. If BPSec only offers data authentication and/or confidentiality, one could ask why the protocol exists, e.g., instead of just applying ciphersuites to Bundle Protocol messages as needed. This question points towards the subtlety of BPSec, namely that it aims to achieve these properties among potentially several corresponding members even when intermediate nodes can add more data to the transmission and/or strip off corrupted data. The reader can here think of information distribution among satellites, where intermediate satellites must relay data or even add to a message, while it is also

possible for data to be corrupted in transit, necessitating partial message discardment to preserve bandwidth. BPSec thus aims to define a channel security protocol that can modularly add and remove data, even while achieving its security goals.

1.1 BPSec: A Flexible Secure Channel

Typically, a secure channel is formed between two communication parties who have previously established a shared key. The security provided by a secure channel construction predominantly focuses on the *confidentiality* and *integrity* of transmitted data, i.e. only the sender and their respective receiver should be able to read and modify the data transmitted. To this extent, there is a rich body of work that formally captures the notions of secure channels as standalone primitives. The seminal works of [BKN02] and [Rog02] introduced the notions of *stateful authenticated encryption* and *authenticated encryption with associated data*, respectively, formalizing early ideas on secure channels. Protections against message replays, reordering, and dropping were early features in [BKN02]. More recently, work has emphasized and differentiated the nature of data transmitted within secure channels, between fragmented streams of data and atomic messages, and formalized the notion of stream-based secure channels [FGMP15]; hierarchies of how channel AEAD notions relate [BHMS16]; and multi-key security [GM17], enabling the analysis of secure channel protocols such as TLS 1.3 [Res18]. The TLS protocol has in fact played a key role in several works, motivating a better understanding of secure channels, with related works spanning robustness [FGJ24], channel termination [BH17], and alternative channel security notions [BMM+15].

While BPSec is a secure channel that aims to establish confidentiality and integrity of transmitted data, modelling BPSec within any existing model for secure channels is difficult due to its unique construction. A typical BPSec node simultaneously maintains a set of shared keys with multiple parties, including the source and destination of a bundle, as well as any intermediate node that might process the bundle on its way towards the final destination. Moreover, unlike a typical message transmitted within a secure channel, the length of a BPSec bundle may be constantly changing, which is an inherent part of its construction. Intermediate nodes that process a bundle may add or remove layers of security from a bundle as per their local policies for processing. This layering of security may sound similar to the design of Tor onion routing [SDM04] but we highlight that these two protocols are strikingly dissimilar for various reasons; unlike a Tor circuit, BPSec paths cannot be pre-established and are subject to change hop-by-hop; message re-encryption is strictly prohibited in BPSec; BPSec integrity checking can be performed on a hop-by-hop basis and is not necessarily end-to-end; and anonymity is not a property captured within the BPSec design. Furthermore, BPSec allows for the partial processing of bundles, i.e., intermediaries may only process blocks from a bundle for which they share a key with the respective source of that block, which may be the bundle source or an intermediate node. This rather "flexible" secure channel construction of BPSec cannot be successfully captured within the rigid formalisms of any existing secure channel frameworks. Thus, we introduce a novel Flexible Secure Channels (FSC) security definition that is capable of capturing the unique nature of security goals for the BPSec protocol: confidentiality and integrity guarantees of individual message blocks within a bundle. We note that our formalism is "flexible" both in the sense that it captures the fluid nature of BPSec security but can also be easily generalized to analyze the security of any channel protocol.

1.2 BPSec Overview

In this section we restate and clarify core components of BPSec RFC 9172 [BM22] and the underlying Bundle Protocol version 7 RFC 9171 [BFB22]. This is a high-level description;

a detailed algorithmic description of BPSec is presented later in Figure 7 which is based upon the mandatory minimum Default Security Context specified in [BWH22].

Entities in a BPSec Channel The *Bundle Protocol Agent* (BPA) is described as a node component that offers the Bundle Protocol services and executes its procedures. We employ a slight abstraction of this for simplicity, and refer to the BPA as the node itself. Possible BPAs may include the *Source Node* (SN) that is the originator of the bundle and the *Destination Node* (DN) which is the intended recipient. *Intermediate Nodes* (IN) may also receive and process bundles – potentially adding or dropping component blocks – before forwarding the transmission. If a BPA $\in$ {SN, IN} adds a security block to a bundle (i.e., that it adds encryption or authentication) it is also called a *security source*. Similarly, if a BPA $\in$ {IN, DN} removes a security block (e.g. decrypting a BCB target) it is also called a *security acceptor*. If the same BPA does not remove the block (e.g. verifying the MAC of a BIB), it is called a *security verifier*.

We give the high-level components of a bundle below:

- *Primary Block* The primary block carries information about the SN, DN, and lifespan of the bundle among other basic bundle identification and forwarding values. There is exactly one primary block per bundle, followed by possible extension blocks and finally a payload block.
- *Extension Block* An extension block provides additional functionality for bundle processing through annotative information (e.g. previous node block, bundle age block, abstract security block, etc.). BIB and BCB are extension blocks that have identical structures as abstract security blocks – both delineating ciphers, parameters, etc. for the target blocks they apply to. INs may also add extension blocks.
- *Payload Block*: There is one payload bock per bundle, always the last block in the bundle, which carries the application data. Since the Bundle Protocol can be used as an encapsulating protocol for another protocol (e.g., an application layer protocol and data) a payload block may be a *partial payload* or *fragmented payload* containing only a segment of the overall payload.
- *Target Block*: The block within a bundle to which a security operation is applied.
- *Security Context*: A security context includes assumptions, algorithms, configurations, and policies that are used to implement security. [1]

Important BPSec Design Decisions through Example BPSec focuses on *block-level granularity and interactions*. Security operations are applied to individual blocks within a bundle according to the security context of a BPA. In this way INs between a SN and DN can add security operations just like a security source, or indeed remove security as if they were an acceptor for the bundle. Figure 2 shows an IN, Charlie, adding BCBs to the bundle it received, acting as a relay from Alice to Bob. While Alice only provided authentication to Block 2 and 4 (creating a BIB block 1 that contains the applicable MAC tags and algorithm information), Charlie decides to AEAD-encrypt certain blocks as well, namely *target blocks* 1, 2, and 4. These are Alice's original blocks including the BIB itself. We also demonstrate a separate BCB in Block 6 that targets only Block 3. It is not consolidated with BCB Block 5 because it uses a different parameter-set. Moreover, this AEAD encryption may have been realized with an encrypt-then-MAC mode vs e.g. GCM; in such cases, BPSec requires the MAC tag output from the BCB to be placed in the BCB block vs being considered as part of the ciphertext in Block 3.

If Charlie shares the same security context and keys with Alice and Bob, then Bob will be able to decrypt and verify the bundle (if the bundle is delivered honestly). Otherwise, if the DN Bob does not have the requisite keys, at least one other IN will be needed to

[1] Security contexts are user-defined but RFC 9173 specifies a mandatory-to-implement security context.

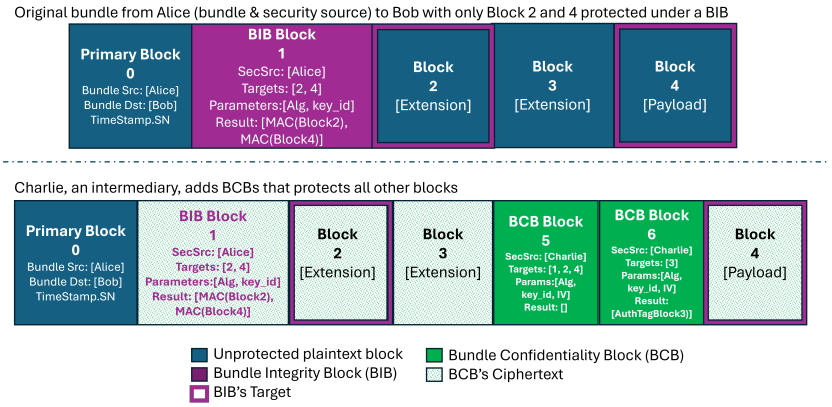

Figure 2: BPSec example showing block interactions between BCB and BIB made by different BPA security sources, Alice and Charlie, for a bundle intended for Bob.

process the BCB blocks for Bob, i.e., to act as a *security acceptor*. Bundle encapsulation, whereby a bundle is encapsulated as a payload of a wrapping bundle, is recommended when the bundle may arrive at the DN before being processed by a security acceptor.

1.3 BPSec Challenges

We note that the DTN threat model gives an attacker complete network access, affording them read/write access to bundles traversing the network. Eavesdropping, modification, topological, and injection attacks are all described in [BM22, Section 8]. Therein, these "on-path attackers" can be unprivileged, legitimate, or privileged nodes depending on their access to cryptographic material: *unprivileged* nodes only have access to publicly shared information, *legitimate* nodes have additional access to keys provisioned for itself, and *privileged* nodes have further access to keys (privately) provisioned for others. In our model, within the symmetric key based Default Security Context [BWH22], all BPSec participants are privileged while attackers may be unprivileged or privileged. There are no guarantees against privileged attacks. In an effort to distinguish malice by INs, we further abstract these classes into *honest* and *dishonest* nodes in our analysis. Honest INs are privileged nodes that faithfully execute the role of a BPA as described in Section 1.2. Dishonest INs are unprivileged nodes that attempt to violate the integrity or confidentiality of blocks it processes (e.g. by dropping or modifying blocks), and captured by our adversary model. We observe a specific gap in guarantees BPSec provides through the BIB, BCB, and the default security context. Specifically, BPSec has no cryptographic auditing mechanism for detecting unprivileged modifications to a bundle between the SN and DN.

Claim: BPSec protections against plaintext modification are insufficient and can lead to a self-imposed denial-of-service. A similar argument can be made for authenticated encrypted BCB target blocks.

- Suppose an unprivileged dishonest IN strips the BIBs and/or BCBs from all bundles it receives and forwards and appends an additional bit to all their associated security targets. BPSec has no in-band mechanisms (i.e. cryptographically enforced) to detect or correct this kind of modification (aside from encrypting the BIB targets after they are authenticated). According to [BM22, 5.1.2], other BPAs who receive this stripped bundle will use an out-of-band security policy mechanism to determine whether to drop, modify, or forward the bundle. Under the recommended security policy, BPAs will remove all target blocks that were supposed to be protected by a BIB. This could lead to dropping the entire bundle if the security policy specified that the primary block must be BIB protected.

- This paradigm sacrifices availability over authenticity. One could argue that availability was never guaranteed by BP and that it is out of scope of BPSec to prioritize availability over authenticity (resp. confidentiality) [2].
- Under a unprivileged dishonest attacker model the DN would not have a means to detect dropped target blocks. This may lead to the DN incorrectly assuming that they have a complete message and acting on it, even if core actionable information was in the dropped blocks.

The core impact of the issues highlighted above is that the destination BPA is unaware of what messages have been removed by the intermediate nodes. To address this, we provide a strong BPSec variant StrongBPSec that offers a *ledger* and *read receipts*. Assuming that SNs always add a *ledger* block and that DNs will not accept a bundle without one, read receipts are designed to provide a verifiable record for processed blocks. This ensures transparency while adding an additional layer of integrity protection between SN and DN within BPSec's symmetric key constraints. The maintenance of a *ledger* block through read-receipts guarantees the integrity of the original bundle created by SN, while simultaneously permitting honest intermediaries to process and discard SN-created security blocks. This should not be construed as the "return-receipt" from a Bundle Status Report [TBW+07] which is an out-of-band (i.e. separate from the bundle) policy-driven acknowledgment of receipt or change status. In fact, an on-path-attacker can simply drop all Bundle Status Reports whereas StrongBPSec offers a stronger in-band (i.e. to the bundle itself) cryptographically enforced audit log.

1.4 Contributions

As can be seen, the BPSec secure channel environment, security expectations, and functionality are quite unlike traditional secure channel protocols, creating a non-trivial environment for analysis. This work formalizes the complex goals of BPSec, provides an analysis of the protocol. This includes a model and proof corresponding to the type of security BPSec can be claimed to offer. We note issues in BPSec and outline normative security goals that it cannot assure. Furthermore, we offer recommendations to enhance security guarantees within the intended design constraints of BPSec by introducing StrongBPSec, which reinforces the integrity assurances of BPSec. StrongBPSec accomplishes this by maintaining a verifiable ledger of changes, allowing the intended recipient of a bundle to independently verify the integrity of modifications made throughout a bundle's journey. We also present a stronger model and analysis to match the improved security of StrongBPSec.

2 Related Work

We first summarize other related work on DTN protocols, BPSec security, and relevant security approaches. Standard definitions that we use in this work for e.g., AEAD and MAC security can be found in Appendix A.

DTN Protocols A variety of DTN protocols for space communications exist and are managed by the Consultative Committee for Space Data Systems (CCSDS) [fSDS23], including the Space Packet Protocol [fSDS20b] and CCSDS File Delivery Protocol (CFDP) [fSDS20a], but relatively little cryptographic analysis has been done of them. The suite of DTN protocols are focused on a store-and-forward approach to make them more robust to environmental disruptions and message relay issues. Since ground stations may also require

[2] We also note that, if only a basic integrity check – not authenticity – of the data is required, the use of the BIB to provide integrity protection is unnecessary as cyclic redundancy check (CRC) codes already exist in BP blocks. CRCs do not provide authenticity.

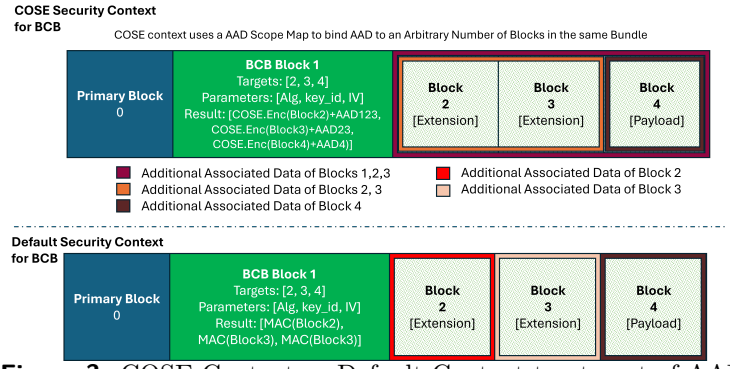

Figure 3: COSE Context vs Default Context treatment of AAD

substantive planning in the order of days to send messages [Man23a], it is essential that each transmission has the capability of aggregating message information along its path and processing as needed. Space link efficiency is a particular concern for DTN protocols.

A survey of DTN key management protocols [MKK17] reveals that formal cryptographic analysis in the provable security sense is relatively rare. Of the ones that have received analysis in the areas of identity based cryptography, non-interactive key exchange, and group key management [AKG+07, KZH07, RSKW17, LML14, ACSA11, ZSS+14], none are documented or promulgated by public institutions such as the IETF or CCSDS as deployed in practice, including in space communications. Other DTN protocols that have known space uses, such as the Licklider Transmission Protocol, lack cryptographic security analysis [BFR08]. Broadly speaking, the lack of cryptographic formal or computational analysis in this field leaves claims of security by various DTN protocols inconclusive, threat models under-defined, and security assumptions on the underlying cryptographic primitives unknown. This presents a gap that we address in this work, providing a first rigorous cryptographic analysis of BPSec.

BPSec COSE Context BPSec's Default Security Context described in [BWH22] was created for interoperability purposes (among space agencies), and provides the minimum level of security based on preshared symmetric keys. In an effort to interoperate with other networks using DTN protocols and achieve compatibility with asymmetric-keyed algorithms, the CBOR Object Signing and Encryption (COSE) Security context was proposed in the DTN IETF working group [Sip24]. This draft standard defines how to incorporate signing and encryption to BPSec using PKI; expands the additional associated data (AAD) to support binding AAD to an arbitrary number of blocks in the same bundle; and introduces PKIX certificate for entity authentication.

Key Wrapping in BPSec BPSec [BWH22] offers the option to incorporate AES Key Wrapping (AESKW) in security blocks, according to the AESKW standards outlined in [HS02]. In an early request for proposal [Dwo04] for ANS X9.102 standard that discuss *key-wrapping* as a primitive, the goal of key wrapping was highlighted as "to protect the confidentiality and integrity of cryptographic keys without the use of nonces". Accordingly, BPSec's use of AESKW aims to enable secure sharing of cryptographic keys used within BIB/BCBs with other nodes who have access to the correct key wrapping keys.

The AESKW wrapping algorithm takes as input a pre-established key encryption key (KEK), a message to encrypt which in this case is a key k, and optional associated data concatenated with a static integrity check vector (ICV) which are passed into a six-round non-standard Feistel-network [RS06]. This outputs a ciphertext which is sent along to a receiver with the plaintext authenticated data. The unwrapping algorithm takes as input

the KEK, ciphertext, ICV and authenticated data and outputs either the shared key or error upon integrity check failure.

A serious caveat with AESWK and other *key-wrap* schemes discussed in ANS X9.102 is that their security has not been formally proven. Rogaway and Shrimpton in [RS06] likens the primitive to a secure enciphering scheme similar to a strong, variable-input-length pseudo-random permutation. However, they stop short of formally verifying the security of AESWK due to ambiguities regarding its construction in ANS X9.102. Instead, they introduce a novel framework called *deterministic authenticated encryption* (DAE) [RS06] that is capable of capturing the security goals of *key-wrapping*, which we leverage in the analysis of BPSec and our StrongBPSec improvements in (Section 5).

Key Management Key management (key derivation, key exchange, key revocation, key separation) policies are not explicitly defined by any of the three relevant RFCs for BPSec. Instead, it is assumed that these are handled separately as part of network management [BM22]. RFC9173 [BWH22] stipulates that BP nodes using security contexts need to "establish shared key encryption keys (KEKs) with other nodes in the network using an out-of-band mechanism". BPSec *does not* provide key establishment or entity authentication mechanisms internally.

Symmetric keys were chosen over asymmetric keys (e.g. BIB with HMAC-SHA2) "in order to create a security context that can be used in all networking environments" [Bir23]. RFC9173 stipulates that different keys must be used to perform different security operations (e.g. a separate key for data encryption vs integrity protection) and across different cipher-suites for the same operations (i.e. using a separate key for AES-GCM vs AES-CBC). Depending on how symmetric keys are distributed for a given security context (key exchange, pre-shared, out-of-band), the use of ephemeral keys through AES-Key wrapping or a key-rotation policy must be used to curtail key leakage. Initialization vectors must also not be reused with the same key across multiple encryption operations.

3 BPSec Formalization and Strong BPSec

We first present a formalization of notation for BPSec, with variables shown in Table 1. A general BPSec block is structured as shown in Figure 4.

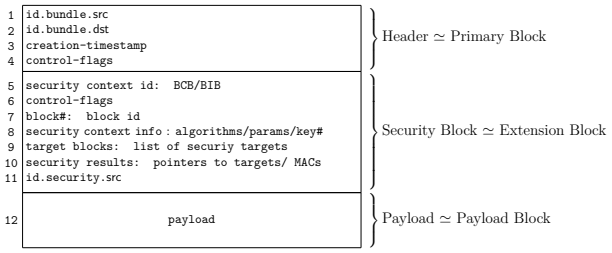

Figure 4: Generic BPSec bundle with cryptographically relevant fields.

We assume a 1:1:1 relationship between $\overrightarrow{\mathtt{M}}, \overrightarrow{\mathtt{F}}, \overrightarrow{\mathtt{P}}$ meaning that for each block in a bundle $\overrightarrow{\mathtt{M}}$, a BPA $\in \{\text{SN, IN}\}$ can add a unique security operation $\overrightarrow{\mathtt{F}}$ governed by a unique parameter set $\overrightarrow{\mathtt{P}}$. The security policy of a BPA may be configured to ignore certain blocks which corresponds to a particular $\overrightarrow{\mathtt{F}}_i = \perp$. Otherwise, the BPA either adds plaintext integrity or authenticated encryption operations to the block yielding in a new BIB or BCB, respectively. We designate the BPSec protected bundle as $\overrightarrow{\mathtt{C}}$. For authenticity protection,

Table 1: Abstracted variables used to formalize BPSec.

$\overrightarrow{\Pi}$	Global state $\{st_1, ..., st_n\}$ that holds particular key sets $\overrightarrow{k}_p$ and parameter sets p such that $\{st_i[p] = (\overrightarrow{k}_p, p)\}$ used by party i participating in a BPSec channel.
aad	Authenticated associated data
bid	Block index which identifies the block written as a subscript of $\overrightarrow{\mathtt{M}}$, $\overrightarrow{\mathtt{F}}$, or $\overrightarrow{\mathtt{P}}$: B - Index of the Strong BPSec ledger block $\mathtt{BIB}_\mathtt{B}$ (always the second to last block in our construction to align with [BFB22]) PB - Index of the Primary Block (always 0) PLD - Index of the bundle payload block (always $\mathtt{B}-1$) tar - Index of security operation's target block
$\overrightarrow{\mathtt{C}}$	Resultant array after operating on $\overrightarrow{\mathtt{M}}$ with a set of cryptographic operations specified by $\overrightarrow{\mathtt{F}}$ and parameterized by $\overrightarrow{\mathtt{P}}$.
c_k	Ciphertext of the key wrapped ephemeral symmetric key k
ctr	Counter for tracking index of security blocks
$\overrightarrow{\mathtt{F}}$	List of security operations $\{\mathsf{conf}, \mathsf{int}\}$ for blocks in a bundle $\overrightarrow{\mathtt{M}}$
$\overrightarrow{\mathtt{F}}.\mathtt{type}$	BPSec block type {"BCB", "BIB"} synonymous with operations $\{\mathsf{conf}, \mathsf{int}\}$
$\overrightarrow{\mathtt{F}}.\mathtt{targets}$	Security operation targets $\{\mathsf{bid}_0, \ldots, \mathsf{bid}_\mathsf{N}\}$
$\overrightarrow{\mathtt{ID}}$	A global set of node identifiers consisting of $\mathtt{id} \in \{\mathtt{id}_1, \ldots, \mathtt{id}_n\}$ to uniquely identify each state st in global state $\overrightarrow{\Pi}$
IV	Initialization vector
k	Ephemeral symmetric key (e.g. the key wrapped key)
k_p	Preshared symmetric key accessible to parties with access to $\overrightarrow{\Pi}$
$k^{\mathtt{BB}}$	Ephemeral symmetric key for $\mathsf{BIB}_\mathtt{B}$ authentication (i.e. the key wrapped $\mathsf{BIB}_\mathtt{B}$ key)
$k_p^{\mathtt{BB}}$	Preshared symmetric key for $\mathsf{BIB}_\mathtt{B}$ authentication
$k^{\mathtt{RR}}$	Ephemeral symmetric key for BRB authentication (i.e. the key wrapped BRB key)
$k_p^{\mathtt{RR}}$	Preshared symmetric key for BRB authentication
$\overrightarrow{\mathtt{M}}$	Plaintext blundle made of blocks $\{\mathsf{bid}_0, \ldots, \mathsf{bid}_\mathsf{n}\}$ where $\mathtt{n} = \lvert\overrightarrow{\mathtt{M}}\rvert$ such that $\lvert\overrightarrow{\mathtt{M}}\rvert \geq 1$ (mandatory primary and payload blocks)
$\overrightarrow{\mathtt{M}}_i.m$	The plaintext data in a block $\overrightarrow{\mathtt{M}}_i$
meta	Metadata ledger for all security operations performed by the security source that is authenticated by the $\mathtt{BIB}_\mathtt{B}$
$\overrightarrow{\mathtt{P}}$	Sets of ciphersuite parameters associated with security operations $\overrightarrow{\mathtt{F}}$ specified by a security context.
$\overrightarrow{\mathtt{P}}.\mathtt{alg}$	Algorithms specified (e.g. AEAD modes, key generation) for a block.
$\overrightarrow{\mathtt{P}}.\mathtt{id}$	Node identifier for a party that supports a particular parameter set
$\overrightarrow{\mathtt{P}}.\mathtt{init}$	T/F flag: T if the security source is the bundle source; F otherwise.
$\overrightarrow{\mathtt{P}}.\mathtt{params}$	Security parameters used for particular algorithms.
$\overrightarrow{\mathtt{P}}.\mathtt{params.kid}$	Key identifier
st	The local state of a party within global state $\overrightarrow{\Pi}$
$st.\mathtt{id}$	The local state for party id
tar	The target block index: $\overrightarrow{\mathtt{M}}_{\mathtt{tar}}$ is the unprotected target block whereas $\overrightarrow{\mathtt{C}}_{\mathtt{tar}}$ is the protected target block.
valid	A helper function to check if the security operation $\overrightarrow{\mathtt{F}}_i$ conflicts with any existing security operations already applied to a block $\overrightarrow{\mathtt{M}}_i$.

the BPA performs HMAC operations using a secret key to the block and stores the tag, a security result in the BIB. For authenticated encryption, AEAD is applied to the block yielding either separate MAC and ciphertext or single combined result: in cases where the MAC is generated separately, it can be stored in the BCB as a result while the ciphertext replaces the data being encrypted in-place. When multiple blocks are protected by the same security operation using the same parameter set, they are consolidated under a single BIB or BCB respectively as seen in Block 5 in Figure 2.

3.1 Flexible Secure Channels

A Flexible Secure Channel (FSC) is distinguished by several features from a traditional secure channel. Specifically, an FSC includes not only a sender and receiver but also one or more intermediate nodes which share a common set of keys $\overrightarrow{k}$ and associated parameters $p \in \overrightarrow{\mathcal{P}}$ contained in their individual *states*. For BPSec, the keys are preinstalled symmetric keys and the parameters are dictated by the security context(s) supported by the participant. The FSC global state $\overrightarrow{\Pi}$ is the union of all local states, st of FSC participants (i.e. $st \in \overrightarrow{\Pi}$). Without loss of generality, in a symmetric key FSC, such as

in BPSec under the Default Security Context, sender state st and receiver state st' must match (i.e. there exists $p \in \overrightarrow{\mathcal{P}}$ such that $st[p] = st'[p]$) to correctly process FSC messages.

Definition 1 (Syntax for flexible secure channels)**.** A flexible secure channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Snd}, \mathsf{Rcv})$ with associated state space $\mathcal{S}$ and error space $\mathcal{E}$, where $\mathcal{E} \cap \{0,1\}^* = \emptyset$, consists of three efficient algorithms:

- Init. On input of a security parameter array $\overrightarrow{\mathtt{P}}$, this probabilistic algorithm outputs initial state array $\overrightarrow{\Pi} \in (\mathcal{S} \times \ldots \times \mathcal{S})$. We write $(\overrightarrow{\Pi}) \xleftarrow{\$} \mathsf{Ch.Init}(\overrightarrow{\mathtt{P}})$. We note that global state $\overrightarrow{\Pi}$ constitute many states st, where a state st matches with multiple other states $st' \in \{st_1, \ldots, st_n\}$.
- Snd. On input of state $st \in \overrightarrow{\Pi}$, a message bundle $\overrightarrow{\mathtt{M}} : |\overrightarrow{\mathtt{M}}| \in \mathbb{N}$ and $\forall m \in \overrightarrow{\mathtt{M}}, m \in \{0,1\}^*$, a security flag array $\overrightarrow{\mathtt{F}}$, and a security parameter array $\overrightarrow{\mathtt{P}}$, this (possibly) probabilistic algorithm outputs an updated state array $st' \in \overrightarrow{\Pi}$ and a secured bundle $\overrightarrow{\mathtt{C}} : |\overrightarrow{\mathtt{C}}| \in \mathbb{N}$ and $\forall c \in \overrightarrow{\mathtt{C}}, c \in \{0,1\}^*$. We write $(st', \overrightarrow{\mathtt{C}} \xleftarrow{\$} \mathsf{Snd}(st, \overrightarrow{\mathtt{M}}, \overrightarrow{\mathtt{F}}, \overrightarrow{\mathtt{P}}))$.
- Rcv. On input of a state $st \in \overrightarrow{\Pi}$ and a secured bundle $\overrightarrow{\mathtt{C}}$, this deterministic algorithm outputs an updated state $st' \in \overrightarrow{\Pi}$ and a message bundle $\overrightarrow{\mathtt{M}} : \forall\ m \in \overrightarrow{\mathtt{M}}$ where $m \in (\{0,1\}^* \cup \mathcal{E})$. We write $(st', \overrightarrow{\mathtt{M}} \leftarrow \mathsf{Rcv}(st, \overrightarrow{\mathtt{C}}))$.

Definition 2 (Flexible secure channel correctness)**.** Let $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Snd}, \mathsf{Rcv})$ be a flexible secure channel. We say that Ch provides correctness if for all state pairs $(st_i, st_j) \in \overrightarrow{\Pi} \xleftarrow{\$} \mathsf{Ch.Init}(\overrightarrow{\mathtt{P}})$, for all messages $(m_0, \ldots, m_\ell) : \overrightarrow{\mathtt{M}} = \{m_0, \ldots, m_\ell\}$ (where $\ell \in \mathbb{N}$) , for all flags $(f_0, \ldots, f_\ell) : \overrightarrow{\mathtt{F}} = \{f_0, \ldots, f_\ell\}$, for all parameters $(p_0, \ldots, p_\ell) : \overrightarrow{\mathtt{P}} = \{p_0, \ldots, p_\ell\}$ and for all message arrays $\overrightarrow{\mathtt{M}}' \leftarrow \mathsf{Rcv}(st_j, \mathsf{Snd}(st_i, \overrightarrow{\mathtt{M}}, \overrightarrow{\mathtt{F}}, \overrightarrow{\mathtt{P}}))$, we have that

$$(m_0, \ldots, m_\ell) \in \overrightarrow{\mathtt{M}}' \iff \forall p_r \in \{p_0, \ldots, p_\ell\} : i, j \in p_r.id \quad .$$

Thus, for a block to be processed correctly by a receiver, the receiver must support the particular parameter set embedded in the associated states for each of the message blocks in the bundle. Additionally, since an intermediate receiving node may not share a matching state (with appropriate parameter sets) for all security blocks in a bundle $\overrightarrow{\mathtt{C}}$, the node may legitimately process only the blocks for which it shares the correct state. This *partial processing* is additionally captured as correct behavior via our definition.

In our protocol construction for BPSec and StrongBPSec illustrated in Figure 7 we leverage the flexibility and modularity of our formalism for flexible secure channels. Notably, within the BPSec construction, the intermediary nodes process bundles differently to source and destination nodes, who only Snd and Rcv bundles respectively. In contrast, intermediary nodes both Snd and Rcv bundles; they Snd as they add new BCB/BIB blocks to a bundle and forward it to the next node; they Rcv bundles forwarded to them within which they process and discard BIB/BCB blocks and add BRB read receipts. The versatility of our flexible channel design successfully captures all these use cases, enabling the formalization of uncommon constructions such as BPSec.

We next provide an intuition for StrongBPSec, before formalizing the construction of both BPSec and StrongBPSec, with a side-by-side clarification of differences, in Figure 7.

3.2 StrongBPSec with Read Receipts

As noted, the current BPsec specification [BM22] provides no security guarantees against arbitrary message dropping by an attacker (privileged or not) during the transit of a bundle between its source and the final destination. Particularly, an inherent feature of BPSec is

to allow honest intermediary nodes to process and discard bundle blocks as governed by their internal policies with the exception of primary and payload blocks. However, BPSec [BM22] does not require intermediate nodes to provide verifiable evidence of identity in order to process or discard security blocks. Neither does the bundle maintain a record of changes made to it on way to its destination. Thus, an unprivileged attacker can arbitrarily and selectively drop blocks from a bundle en route without being detected. The proposed COSE Security Context for BPSec [Sip24], as part of its integration into BPSec, includes a recommendation to increase the integrity of bundle blocks. In order to provide stronger integrity guarantees, they propose binding an arbitrary number of blocks together in the same bundle, using additional associated data (`aad`) for each block concerned. However, we argue that binding `aad` to an arbitrary number of blocks in this manner also restricts the ability of an honest intermediary to process and discard any individual block from such a group. In an attempt to find a fair middle ground between the somewhat relaxed original functionality goals of BPSec [BM22] and the overly rigid security guarantees of COSE[Sip24], we propose our StrongBPSec protocol with *read receipts*.

StrongBPSec introduces two additional layers of integrity that aims to guarantee to the bundle destination one of two of the following:

1. All blocks added by the bundle source node SN has arrived unchanged at the bundle destination node DN, or
2. Some honest intermediary node IN has correctly processed and discarded block/s that were constructed by the bundle source SN.

Note that, for clarity, we will be exclusively distinguishing the roles of BPA as either SN, IN or DN for the rest of this discussion.

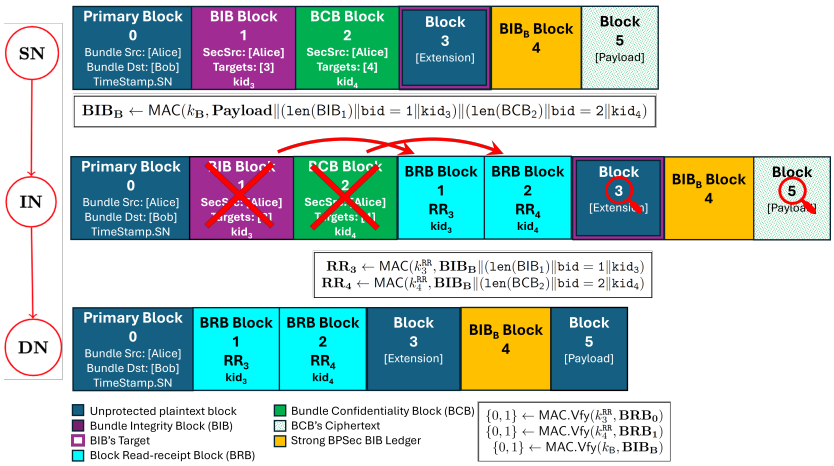

Figure 5: StrongBPSec with Read Receipts. Source integrity block MAC $\mathsf{BIB_B}$ is calculated over the payload and metadata for each target/security block ($\mathsf{BIB_1}$ and $\mathsf{BCB_2}$) added by SN. Read receipts BRB are calculated by INs every time they successfully process a security block added by SN, recording and verifying the details of the blocks they process. INs replace any such security blocks they process with a corresponding BRB before forwarding the bundle. DN first verifies any BRBs added by INs followed by the verification of $\mathsf{BIB_B}$.

We achieve these stronger integrity guarantees through the addition of two additional checks leveraging the already existing BIB format of BPSec. As illustrated in Figure 5, we introduce a $\mathsf{BIB_B}$ block, calculated and prepended to the payload block by the original source node SN. As the originator of the bundle, SN constructs $\mathsf{BIB_B}$, which includes a MAC tag calculated over the payload block and a set of uniquely identifying details

for each security block added by SN. These identifying details include unique key/block identifiers (kid/bid), and the number of target blocks len(BIB/BCB) covered by a security block (BIB/BCB). Once constructed, this BIB_B is only verified by the destination node DN. [3] The successful verification of BIB_B with the corresponding key k^{BB}, which is either a pre-shared or a key-wrapped key, informs DN one of two things; either they have received the exact same bundle SN constructed; or an honest intermediary IN has processed and discarded some of the blocks but have replaced them with correct read receipts. These read receipt blocks duplicate the unique identifying details of a discarded block authenticated with MAC tag, which we discuss next. In line with the BPSec specification, we assume honest privileged intermediary nodes.

The second type of integrity check introduced by our StrongBPSec construction is the concept of a *read receipt* or Block Read-receipt Block (BRB). The addition of a BRB allows an honest intermediary acting as a security acceptor to process and discard any block added by SN but still maintain a verifiable record of their details that was used in the calculation of BIB_B. To clarify, recall from Section 1.3, a read receipt BRB provides an in-band ledger of changes for processing at the DN; it should not be assumed that the BRB gets sent back to the SN as some sort of acknowledgment of receipt or changes as would be done by a "return-receipt" from a Bundle Status Report [TBW+07].

A BRB contains a plaintext and a MAC tag calculated over it. The plaintext duplicates the identifying details of the corresponding discarded security block (kid/bid, len(BIB/BCB)) that was used in the calculation of BIB_B, without which the verification of BIB_B at DN will fail. A MAC tag is then calculated using a unique key k^{RR}, which is either a pre-shared or a key-wrapped key, over the bundle BIB_B concatenated with this plaintext. These BRBs act as a transcript of the original bundle to the DN, who verifies all BRBs prior to the verification of BIB_B. The successful verification of BRBs within a received bundle informs DN that only honest intermediaries have discarded blocks from the original bundle. In Figure 6 we illustrate an expected execution of our StrongBPSec formalism, which we describe in detail in Figure 7.

While significant security gains could be achieved with inclusion of digital signatures in this improved protocol (in particular preventing impersonation within a group sharing the same parameter set and keys), the BPSec infrastructure does not assume asymmetric key management; pre-shared symmetric keys were favored for simplicity and broad implementation. Thus, in strengthening BPSec, we inherit the original key infrastructure and primitives assumed by the Default Security Context described in RFC9173 [BWH22]. Thus, it is not possible to detect which IN has edited the bundle nor is it possible to protect against an insider threat. These issues are inherent to RFC9173 in absence of asymmetric key management.

Figure 7 shows the formal construction of BPSec under RFC9173 alongside our StrongBPSec. The construction abstracts details of BPSec [BM22] and its Default Security Context [BWH22] into a FSC protocol (see Definition 1) with modifications needed for a StrongBPSec protocol. Below we explain the intuition behind our formal construction.

- Init generates symmetric keys for matching BPSec states per parameter set. Parameters in a state are associated with a collection of party identifiers $\vec{\mathtt{id}}$ that all maintain the same symmetric keys. Init generates three keys per parameter set: a symmetric key k_p; a read receipt key k_p^{RR} and; a bundle verifier key k_p^{BB}.
- Snd creates security blocks either by bundle source or any intermediate node. For each message $m \in \vec{\mathtt{M}}$, Snd checks that the security operation is valid, and key wraps a new symmetric key if necessary. Snd adds the appropriate processing information to the security block, and either authenticates or encrypts the message m according

[3] We note that we only consider the use case for DN for simplicity but WLOG we say that any *security acceptor* authorized by the security policy may process BIB_B.

to the security operation. In StrongBPSec, Snd also adds a meta-data array for its ledger block, which it authenticates, creating a verifiable ledger.

- Rcv processes security blocks either by bundle destination or any intermediate node. For each ciphertext $c \in \vec{C}$, Rcv checks that the security acceptor has the correct keys to process the ciphertext, and key-wraps a new read-receipt key if necessary as intermediate node. Rcv adds read-receipts after processing the security block, and authenticates the associated meta-data. In StrongBPSec, the destination node also verifies all read-receipts, and constructs a meta-data array for the ledger block and verifies the BIB_B, rejecting the payload if any checks fail. Relevant keys are either extracted from state or decrypted from *keywrapped* ciphertexts.

We formally prove the respective security of BPSec and StrongBPSec in Section 5.

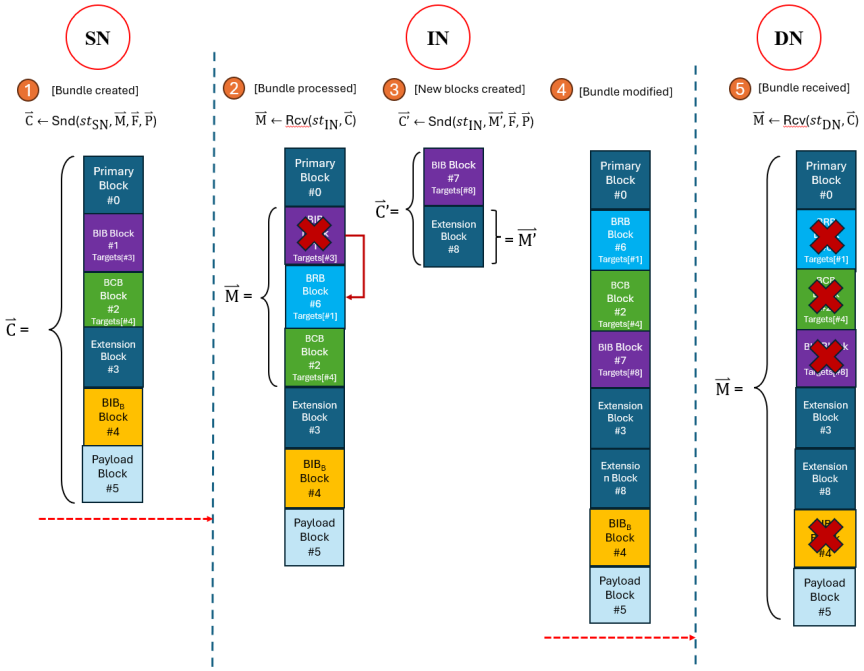

Figure 6: Flow of interactions during an expected execution of StrongBPSec. SN creates a bundle and forwards it to IN. IN processes and replaces BIB block #1 with read receipt block BRB #6. IN further adds new blocks #7 and #8 and forwards the modified bundle to DN. DN processes and discards all remaining security blocks and extracts the bundle message $\vec{M}$. All processed and discarded blocks are crossed out in red. The values for $\vec{P}$ and $\vec{F}$ are selected from respective sets which can be found in Table 1.

4 Flexible Secure Channel Models

In Figure 8 we formalize the security experiment for Flexible Secure Channels (FSC) that captures the BPSec security goals described by [BM22]. Furthermore, in Figure 8 we also capture the intended security goals added by StrongBPSec, namely through Strong FSC Security (SFSC) security, highlighting these as additional steps.

In the FSC experiment, the challenger begins by generating the full secret state for all parties, and randomly sampling a bit b. The adversary is then split into two phases. First, in the Corrupt phase, the $\mathcal{A}$ is allowed to issue Corrupt queries for particular parameter

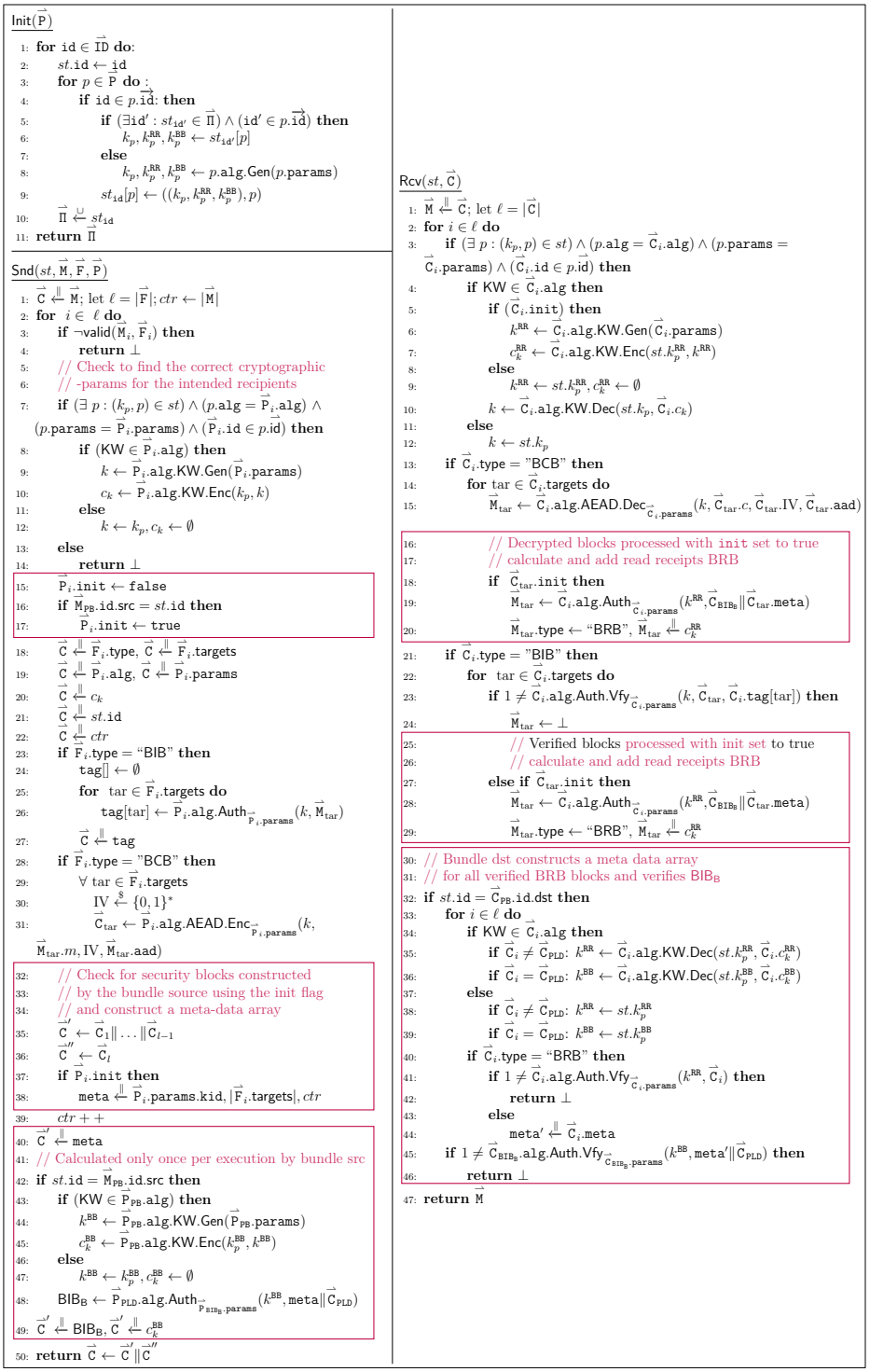

Figure 7: BPSec protocol construction. The additions needed for the StrongBPSec protocol are highlighted in boxes. Refer to Table 1 for notational and functional definitions. Additional notational clarifications: $X \xleftarrow{\|} Y$ denotes concatenating Y to X; $T[j]$ denotes the value accessed by key j in table T, which we assign no particular structure; subscripts are generally used for array indices or but are also used to associate variables (e.g. k_p to denote a key k associated with p) or to configure a function call (e.g. $\mathsf{Auth}_{\vec{\mathtt{C}}_i.\mathtt{params}}$); superscripts are used to classify a variable type (e.g. k^{RR} refers to a read-receipt key); $p.\mathtt{alg}.\mathsf{Gen}$ refers to invoking a key generation function from a family of algorithms such as Key-Wrap (KW) in p but we also overload `alg` as an identifier (e.g. used for boolean comparison).

sets[4]. In the next phase, $\mathcal{A}$ outputs some state, which is given as input to the next adversary $\mathcal{A}'$, which allows us to capture static corruption adversaries.

We argue that proving the security of our construction against a static (as opposed to adaptive) $\mathcal{A}$ is sufficient for the following reasons: each BPSec block is associated with a unique set of independent security parameters that remain unchanged for the duration of the message lifetime; there is no forward secrecy and it does not make a difference at what point of a message's lifetime $\mathcal{A}$ compromises these security parameters as they do not update; and adversary $\mathcal{A}$ compromising of set of security parameters for a specific block does not affect the security of any other blocks in the same bundle. $\mathcal{A}$ now has access to two oracle queries: $\mathcal{O}\mathsf{Snd}$ and $\mathcal{O}\mathsf{Rcv}$:

- $\mathcal{O}\mathsf{Snd}(i, \overrightarrow{\mathtt{M}}, \overrightarrow{\mathtt{F}}, \overrightarrow{\mathtt{P}}) \rightarrow \overrightarrow{\mathtt{C}}$: allows the adversary to indicate that party i should protect the plaintext bundle $\overrightarrow{\mathtt{M}}$ using security operations $\overrightarrow{\mathtt{F}}$ using security parameters $\overrightarrow{\mathtt{P}}$. If the bit b sampled by the challenger is 1, the associated parameters have not been corrupted, and the security operation is encryption (signified by $\mathsf{type} = \mathsf{BCB}$) then the challenger records and replaces the associated ciphertext with random strings from the same length. $\mathcal{O}\mathsf{Snd}$ also records all integrity protected blocks (signified by $\mathsf{type} = \mathsf{BIB}$), and their associated tags in the AUTH register (to detect integrity wins later in the experiment).
- $\mathcal{O}\mathsf{Rcv}(i, \overrightarrow{\mathtt{C}}) \rightarrow \overrightarrow{\mathtt{M}}$: allows the adversary to indicate that party i should process the ciphertext $\overrightarrow{\mathtt{C}}$. $\mathcal{O}\mathsf{Rcv}$ also checks if the adversary has caused any win events to trigger. Specifically, if the associated security parameters are not corrupted, and the party i outputs a valid block which no honest party produced, then the adversary has forged this block, and the challenger sets $\mathtt{win} \leftarrow \mathsf{true}$.

At some point the adversary terminates and outputs a bit b', and the output of the experiment is $(b = b') \vee \mathtt{win}$. We say an adversary $\mathcal{A}$ wins the FSC security game if they succeed in achieving one of the following: forges an integrity-protected block; forges an authenticated ciphertext within a block or; distinguishes the real-or-random ciphertext within a block. Below we briefly summarize additional notations used in our security experiment illustrated in Figure 8 for our security experiment.

- AUTH: A register for honestly authenticated messages and their MAC tags.
- CORR: A register used to track corrupted parameters for determining wins.
- CTXT: A register that maintains either real ciphertext and associated parameter pairs or uniformly random strings from the same space.
- Extr: A helper function to extract a parameter set from a block and state.
- Proc: A binary function that checks if a ciphertext bundle $\overrightarrow{\mathtt{C}}$ can be processed.
- tsid: A state identifier that identifies the parameter set used by source.
- TXT:A register for all messages accessible by the global state $\overrightarrow{\Pi}$.

We now formally define FSC security.

Definition 3 (FSC Security)**.** Let Ch be a channel protocol $\mathsf{Ch} = \{\mathsf{Init}, \mathsf{Snd}, \mathsf{Rcv}\}$. Let $\mathsf{Exp}^{\mathsf{FSC}}_{\mathsf{Ch},\mathcal{A}}(\lambda)$ be the FSC security experiment without red-boxed lines defined in Figure 8. We define the advantage $\mathsf{Adv}^{\mathsf{FSC}}_{\mathsf{Ch},\mathcal{A}}(\lambda)$ that the adversary $\mathcal{A}$ wins the FSC game as $\mathsf{Adv}^{\mathsf{FSC}}_{\mathsf{Ch},\mathcal{A}}(\lambda) = |2 \cdot \Pr(\mathsf{Exp}^{\mathsf{FSC}}_{\mathsf{Ch},\mathcal{A}}(\lambda) = 1) - 1|$. We say that Ch is FSC-secure if $\mathsf{Adv}^{\mathsf{FSC}}_{\mathsf{Ch},\mathcal{A}}(\lambda)$ is negligible in the security parameter λ.

Next, we turn to defining the Strong FSC game (SFSC), which adds the boxes in Figure 8. The SFSC game adds two additional win conditions in the Rcv query, allowing

[4]This describes how $\mathcal{A}$ becomes a dishonest privileged node.

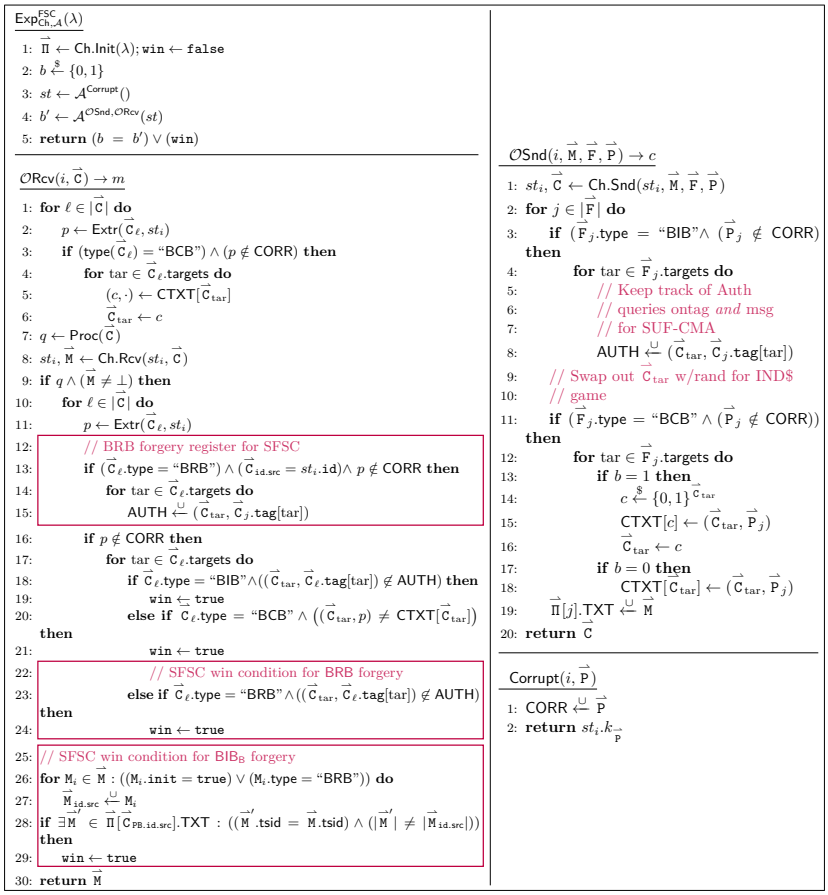

Figure 8: Security Definition for Flexible Secure Channels (FSC). The modifications needed for SFSC experiment for the StrongBPSec with Read Receipts is presented in boxes. Refer to Table 1 and Section 4 for notational definitions.

the adversary to win by dropping $\vec{\mathtt{C}}$ blocks without being detected, or by forging so-called read receipts, which indicate to the receiving party that an honest party has "processed" a block from the sender. As before, the adversary at some point will terminate and outputs a bit b' and the output of the experiment is $(b = b') \vee \mathtt{win}$. Thus, in addition to the FSC win conditions, the SFSC adversary $\mathcal{A}$ wins if it can forge either BRB or BIB_B with non-negligible probability. We now formally define SFSC security.

Definition 4 (SFSC Security)**.** Let Ch be a channel protocol $\mathsf{Ch} = \{\mathsf{Init}, \mathsf{Snd}, \mathsf{Rcv}\}$. Let $\mathsf{Exp}^{\mathsf{SFSC}}_{\mathsf{Ch},\mathcal{A}}(\lambda)$ be the SFSC security experiment defined in Figure 8 (with all lines included). We define the advantage $\mathsf{Adv}^{\mathsf{SFSC}}_{\mathsf{Ch},\mathcal{A}}(\lambda)$ that the adversary $\mathcal{A}$ wins the SFSC game as $\mathsf{Adv}^{\mathsf{SFSC}}_{\mathsf{Ch},\mathcal{A}}(\lambda) = |2 \cdot \Pr(\mathsf{Exp}^{\mathsf{SFSC}}_{\mathsf{Ch},\mathcal{A}}(\lambda) = 1) - 1|$. We say that Ch is SFSC-secure if $\mathsf{Adv}^{\mathsf{SFSC}}_{\mathsf{Ch},\mathcal{A}}(\lambda)$ is negligible in the security parameter λ.

5 Security Analysis

In this section, we provide a security analysis of the BPSec construction given in Figure 7 under Flexible Secure Channel security described in Definition 3. Next, we provide a security analysis of the StrongBPSec construction given in Figure 4 under Strong FSC

security in Definition 4. We begin with our analysis of BPSec.

Theorem 1 (FSC Security for BPSec)**.** *Let n_a be the total number of parameter sets used in the experiment, and let n_m be the total number of key-wraps keys in the experiment. The* BPSec *protocol presented in Figure 7 is* FSC*-secure. That is, for any PPT algorithm $\mathcal{A}$ against the* FSC *security experiment (described in Definition 3)* $\mathsf{Adv}^{\mathsf{FSC}}_{\mathsf{BPSec},\mathcal{A}}(\lambda)$ *is negligible under the* sufcma*,* aead*, and* dae *security of the* MAC*,* AEAD*,* DAE *primitives, respectively.*

Proof. We note that in the base FSC game there are three main ways the $\mathcal{A}$ can win. First, by forging a BIB tag, secondly by forging a BCB ciphertext, and thirdly by guessing the bit b. Thus we divide our proof into three cases: in the first case the adversary has caused $\texttt{win} \leftarrow \texttt{true}$ by generating a forged MAC tag, in the second case the adversary has caused $\texttt{win} \leftarrow \texttt{true}$ by forging an AEAD ciphertext, and finally the third case the adversary guesses the challenger bit b. In each case we upper-bound the probability of the adversary causing the winning event to occur, and thus prove that the BPSec construction is FSC secure. We thus split our analysis into the following cases:

C_1 : $\mathcal{A}$ sets $\texttt{win} \leftarrow \texttt{true}$ when a party π^i accepts $(\overrightarrow{\texttt{M}}'_t, \overrightarrow{\texttt{C}}_j.\texttt{tag}[t])$ such that $\overrightarrow{\texttt{M}}'_t \neq \perp$, but $(\overrightarrow{\texttt{M}}'_t, \overrightarrow{\texttt{C}}_j.\texttt{tag}[t]) \notin \mathsf{AUTH}$ for $\overrightarrow{\texttt{C}}_j.\texttt{type} = \mathsf{BIB}$ and $\mathsf{Corrupt}(\cdot, \overrightarrow{\texttt{C}}_j.\overrightarrow{\texttt{P}})$ was not issued, i.e. $\mathcal{A}$ has successfully forged a valid message MAC tag pair for a BIB that verifies correctly.

C_2: $\mathcal{A}$ sets $\texttt{win} \leftarrow \texttt{true}$ when a party π^i accepts $\overrightarrow{\texttt{C}}$ such that $\overrightarrow{\texttt{M}}_t \neq \perp$, $\overrightarrow{\texttt{C}}_t.\texttt{type} = \mathsf{BCB}$, $p \leftarrow \mathsf{Extr}(\overrightarrow{\texttt{C}}_t, \texttt{st}_i)$ and $\mathsf{Corrupt}(\cdot, \overrightarrow{\texttt{C}}_j, \overrightarrow{\texttt{P}})$ was not issued, i.e. $\mathcal{A}$ has successfully forged a valid AEAD ciphertext.

C_3 $\mathcal{A}$ does not set $\texttt{win} \leftarrow \texttt{true}$, and has terminated the experiment and output a bit b', i.e. $\mathcal{A}$ has guessed the challenge bit b.

We proceed via a sequence of games with each game focusing on a specific win condition (or lack thereof) in the three cases. We bound the difference in the adversary's advantage in each game with the underlying cryptographic assumptions until the adversary reaches a game where the advantage of that game equals 0, which shows that adversary $\mathcal{A}$ cannot win with non-negligible advantage.

Case 1 : π^i **accepts a forged message tag pair for a** BIB.

Game 0 This is the initial FSC security game. Thus $\mathsf{Adv}^{\mathsf{FSC}}_{\mathsf{BPSec},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_0}(\lambda)$.

Game 1 In this game, for any key-wrapping keys associated with parameter sets that have not been Corrupted, we abort if $\mathcal{A}$ forges a DAE ciphertext, and replace honestly generated key-wrapped keys with uniformly random values. To do so, by a hybrid argument we introduce a series of reductions $\mathcal{B}_1$ as follows: At the beginning of the experiment, $\mathcal{B}_1$ will initialise a DAE challenger $\mathcal{C}_{\mathsf{DAE},i}$ for each parameter set $\overrightarrow{\texttt{P}}_i$ such that $\mathsf{Corrupt}(\cdot, \overrightarrow{\texttt{P}}_i)$ has not been issued (where i is the index into the hybrid argument). Whenever the challenger is required to key wrap ephemeral key k' using k_i, the challenger instead queries $(\emptyset, k')$ to the associated $\mathcal{C}_{\mathsf{DAE},i}$ and replaces the honestly generated key-wrap ciphertext with the output from $\mathcal{C}_{\mathsf{DAE},i}$. Additionally, each time $\mathcal{B}_1$ generates a ciphertext from $\mathcal{C}_{\mathsf{DAE},i}$, they maintain a table $\mathsf{DAE}[i] \leftarrow (C, k')$ which they update with each honestly generated key-wrap ciphertext. Whenever $\mathcal{B}_1$ is required to decrypt a DAE ciphertext using the key-wrap key associated with parameter set $\overrightarrow{\texttt{P}}_i$, they first check if $(C, k') \in \mathsf{DAE}[i]$. If so, they use k' as the key-wrapped key. If not, they submit $(\emptyset, C)$ to $\mathcal{C}_{\mathsf{DAE},i}$ challenger.

Note that by definition of the DAE security game, if $\mathcal{A}$ forges a DAE ciphertext and the bit b sampled by DAE is 0, then DAE will return the correct decryption of C, otherwise DAE returns $\perp$. Thus, any adversary that can forge a DAE ciphertext can be used to break the security of the DAE game. Additionally, we note that if the bit b sampled by $\mathcal{C}_{\mathsf{DAE},i}$ is 0, then the $\mathcal{C}_{\mathsf{DAE},i}$ encrypts the ephemeral key-wrapped keys k' honestly, and we are in

Game 0. Otherwise, if the bit b sampled by the $\mathcal{C}_{\mathsf{DAE},i}$ is 1, then $\mathcal{C}_{\mathsf{DAE},i}$ simply samples a ciphertext uniformly at random, and now the key k' is uniformly random and completely independent of the protocol flow and we are in **Game 1**. Any $\mathcal{A}$ that could distinguish this change can be directly used to break the DAE security of the DAE primitive.

We note that $\mathcal{C}_{\mathsf{DAE},i}$ samples keys identically to the security experiment, and thus this replacement is sound. Finally, we note that $\mathcal{A}$ can never later call $\mathsf{Corrupt}(\cdot, \overrightarrow{\mathtt{P}}_i)$, so all internal values to DAE will never need to be revealed. As a result of these changes we know that any key-wrap output from a parameter set that has not been Corrupted is uniformly random and independent of the protocol flow, and $\mathcal{A}$ has no information on this key. There are at most n_a total parameter sets and thus n_a hybrid arguments and thus we find: $\mathsf{Adv}^{\mathcal{A}}_{G_0}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_1}(\lambda) + n_a \cdot \mathsf{Adv}^{\mathsf{dae}}_{\mathsf{DAE},\mathcal{B}_1}(\lambda)$.

Game 2 In this game, we introduce an abort event that triggers if any un-Corrupted party "accepts" any of the MAC tags $\sigma_t \in \overrightarrow{\mathtt{C}}_j.\mathtt{tag}[t]$ for $t \in \overrightarrow{\mathtt{C}}_j.\mathsf{targets}$ of a block $j \in |\overrightarrow{\mathtt{C}}|$ such that $\overrightarrow{\mathtt{C}}_j.\mathsf{type} = \mathsf{BIB}$, $\overrightarrow{\mathtt{M}}_t \neq \bot$, and no honest party generated a message $\overrightarrow{\mathtt{M}}_t$. Specifically, this occurs if $\mathcal{A}$ is able to successfully forge BIB using a key associated with a parameter set $p = \mathsf{Extr}(\overrightarrow{\mathtt{C}}_t, \mathsf{st}_i)$ such that $\mathsf{Corrupt}(\cdot, p)$ was not issued. We do so by introducing a series of reductions $\mathcal{B}_2$ as follows: At the beginning of the experiment, for each parameter set $\overrightarrow{\mathtt{P}}_i$ that does not support key-wrapping $\mathcal{B}_2$ initialises a MAC challenger $\mathcal{C}_{\mathsf{sufcma},i}$. Additionally, during the experiment if a DAE ciphertext is generated or decrypted using k_{p_i} (where $\mathcal{A}$ has not issued $\mathsf{Corrupt}(\cdot, p_i)$), $\mathcal{B}_2$ initialises a MAC challenger $\mathcal{C}_{\mathsf{sufcma},i}$. Next, whenever $\mathcal{B}_2$ is required to generate a MAC tag over a message m using k_{p_i} (or the ephemeral key-wrapped key k' encrypted under k_{p_i}) for p_i, $\mathcal{B}_2$ instead queries m to $\mathcal{C}_{\mathsf{sufcma},i}$. If $\overrightarrow{\mathtt{C}}_t$ verifies correctly but was not produced by an honest security source, then $(\overrightarrow{\mathtt{M}}_t, \sigma_t) \notin \mathsf{AUTH}$ and $\mathcal{A}$ has broken the sufcma security of the MAC. We note that by **Game 1** and the definition of the security experiment, all MAC keys replaced in this way were already uniformly random and independent of the protocol flow, and $\mathcal{A}$ cannot learn these by issuing a Corrupt query. We have at most n_a parameter sets that may be used directly as MAC keys, and at most n_m key-wrapped MAC keys, thus $\mathsf{Adv}^{\mathcal{A}}_{G_1}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_2}(\lambda) + (n_a + n_m) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathcal{B}_2,\mathsf{MAC}}(\lambda)$.

Case 1 Analysis: By definition of **Case 1** we know that $\mathcal{A}$ has caused $\mathtt{win} \leftarrow \mathtt{true}$ by forcing some party π^i to accept a message block $\overrightarrow{\mathtt{M}}_t$ such that $\overrightarrow{\mathtt{C}}_t.\mathsf{type} = \mathsf{BIB}$, $\mathsf{Corrupt}(\cdot, \mathsf{Extr}(\overrightarrow{\mathtt{C}}_t, st_i))$ has not been issued. By **Game 1** we replaced the ephemeral key k' used to generate the BIB if p is a key-wrapping algorithm. By **Game 2** we added an abort query that prevents $\mathcal{A}$ from forging MAC tags and thus BIB blocks. Since we abort if the adversary forges their own valid BIB, by **Game 2** π^i aborts and thus $\mathcal{A}$ cannot win the game. Thus we have: $\mathsf{Adv}^{\mathcal{A}}_{G_2}(\lambda) = 0$, and it follows that $\mathsf{Adv}^{\mathsf{FSC}}_{\mathsf{BPSec},\mathcal{A},C_1}(\lambda) \leq n_a \cdot \mathsf{Adv}^{\mathsf{dae}}_{\mathsf{DAE},\mathcal{B}_1}(\lambda) + (n_a + n_m) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathcal{B}_2,\mathsf{MAC}}(\lambda)$ We now proceed to **Case 2**.

Case 2 : π^i **accepts a forged message, ciphertext pair for a** BCB

Game 0 This is the initial FSC security game. Thus $\mathsf{Adv}^{\mathsf{FSC}}_{\mathsf{BPSec},\mathcal{A},C_2}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_0}(\lambda)$.

Game 1 This game proceeds identically to **Game 1** of **Case 1**, with the same reductions and bounds. Thus we have $\mathsf{Adv}^{\mathcal{A}}_{G_0}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_1}(\lambda) + n_a \cdot \mathsf{Adv}^{\mathsf{dae}}_{\mathsf{DAE},\mathcal{B}_3}(\lambda)$.

Game 2 In this game, we introduce an abort event that triggers if any un-Corrupted party "accepts" a ciphertext block $\overrightarrow{\mathtt{C}}_t$ such that $\overrightarrow{\mathtt{C}}_t.\mathsf{type} = \mathsf{BCB}$, $\overrightarrow{\mathtt{M}}_t \neq \bot$, and no honest party generated a message $\overrightarrow{\mathtt{M}}_t$. Specifically, this occurs if $\mathcal{A}$ is able to successfully forge BCB using a key associated with a parameter set $p = \mathsf{Extr}(\overrightarrow{\mathtt{C}}_t, \mathsf{st}_i)$ such that $\mathsf{Corrupt}(\cdot, p)$ was not issued. We do so by introducing a series of reductions $\mathcal{B}_4$ as follows: At the beginning of the experiment, for each parameter set $\overrightarrow{\mathtt{P}}_i$ that does not support key-wrapping, $\mathcal{B}_4$ initialises an auth-security AEAD challenger $\mathcal{C}_{\mathsf{auth},i}$. Additionally, during the experiment if a DAE

ciphertext is generated or decrypted using k_{p_i} (where $\mathcal{A}$ has not issued $\mathsf{Corrupt}(\cdot, p_i)$), $\mathcal{B}_4$ initialises an auth AEAD challenger $\mathcal{C}_{\mathsf{auth},i}$.

Next, whenever $\mathcal{B}_4$ is required to generate a ciphertext over a message, nonce and header tuple (m, N, H) using k_{p_i} (or the ephemeral key-wrapped key k' encrypted under k_{p_i}) for p_i, $\mathcal{B}_4$ instead queries (m, N, H) to $\mathcal{C}_{\mathsf{auth},i}$'s encryption oracle. Similarly, if $\mathcal{B}_4$ is required to decrypt a ciphertext, nonce, header tuple (C, N, H) for k_{p_i} (or the ephemeral key-wrapped key k' encrypted under k_{p_i}) for parameter set p_i, $\mathcal{B}_4$ instead queries (C, N, H) to $\mathcal{C}_{\mathsf{auth},i}$'s decryption oracle. If $\overrightarrow{\mathsf{C}}_t$ decrypts correctly but C was not produced by an honest security source, then $(\overrightarrow{\mathsf{C}}_t, p_i) \notin \mathsf{CTXT}$ and $\mathcal{A}$ has broken the auth security of the AEAD scheme. Submitting (C, H, N) to $\mathcal{C}_{\mathsf{auth},i}$'s decryption oracle then allows $\mathcal{B}_4$ to win the auth-security game. We note that by **Game 1** and the definition of the security experiment, all AEAD keys replaced in this way were already uniformly random and independent of the protocol flow, and $\mathcal{A}$ cannot learn these by issuing a $\mathsf{Corrupt}$ query. We have at most n_a parameter sets that may be used directly as AEAD keys, and at most n_m key-wrapped AEAD keys, thus $\mathsf{Adv}^{\mathcal{A}}_{G_1}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_2}(\lambda) + (n_a + n_m) \cdot \mathsf{Adv}^{\mathsf{auth}}_{\mathcal{B}_4,\mathsf{AEAD}}(\lambda)$.

Case 2 Analysis: By definition of **Case 2** we know that $\mathcal{A}$ has caused $\mathtt{win} \leftarrow \mathtt{true}$ by forcing some party π^i to accept a message block $\overrightarrow{\mathsf{C}}_t$ such that $\overrightarrow{\mathsf{C}}_t.\mathsf{type} = \mathsf{BCB}$, and $\mathsf{Corrupt}(\cdot, \mathsf{Extr}(\overrightarrow{\mathsf{C}}_t, st_i))$ has not been issued. By **Game 1** we replaced the ephemeral key k' used to generate the BCB if p is a key-wrapping algorithm. By **Game 2** we added an abort query that prevents $\mathcal{A}$ from forging ciphertexts and thus BCB blocks. Since we abort if the adversary forges their own valid BCB, by **Game 2** π^i aborts and thus $\mathcal{A}$ cannot win the game. Thus we have: $\mathsf{Adv}^{\mathcal{A}}_{G_2}(\lambda) = 0$, and it follows that $\mathsf{Adv}^{\mathsf{FSC}}_{\mathsf{BPSec},\mathcal{A},C_2}(\lambda) \leq n_a \cdot \mathsf{Adv}^{\mathsf{dae}}_{\mathsf{DAE},\mathcal{B}_1}(\lambda) + (n_a + n_m) \cdot \mathsf{Adv}^{\mathsf{auth}}_{\mathcal{B}_2,\mathsf{AEAD}}(\lambda)$ We proceed to **Case 3**.

Case 3 : $\mathcal{A}$ has terminated the experiment and output a bit b', i.e. $\mathcal{A}$ has guessed the challenge bit b.

Game 0 This is the initial FSC security game. Thus $\mathsf{Adv}^{\mathsf{FSC}}_{\mathsf{BPSec},\mathcal{A},C_3}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_0}(\lambda)$.

Game 1 This game proceeds identically to **Game 1** of **Case 1**, with the same reductions and bounds. Thus we have $\mathsf{Adv}^{\mathcal{A}}_{G_0}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_1}(\lambda) + n_a \cdot \mathsf{Adv}^{\mathsf{dae}}_{\mathsf{DAE},\mathcal{B}_5}(\lambda)$.

Game 2 In this game, all ciphertexts produced by un-$\mathsf{Corrupt}$ed party generating a ciphertext block $\overrightarrow{\mathsf{C}}_t$ such that $\overrightarrow{\mathsf{C}}_t.\mathsf{type} = \mathsf{BCB}$, using a key associated with a parameter set $p = \mathsf{Extr}(\overrightarrow{\mathsf{C}}_t, \mathsf{st}_i)$ such that $\mathsf{Corrupt}(\cdot, p)$ was not issued are replaced with uniformly random and independent strings. We do so by introducing a series of reductions $\mathcal{B}_6$ as follows: At the beginning of the experiment, for each parameter set $\overrightarrow{\mathsf{P}}_i$ that does not support key-wrapping $\mathcal{B}_6$ initialises an PRIV-security AEAD challenger $\mathcal{C}_{\mathsf{PRIV},i}$. Additionally, during the experiment if a DAE ciphertext is generated or decrypted using k_{p_i} (where $\mathcal{A}$ has not issued $\mathsf{Corrupt}(\cdot, p_i)$), $\mathcal{B}_6$ initialises an PRIV AEAD challenger $\mathcal{C}_{\mathsf{PRIV},i}$.

Next, whenever $\mathcal{B}_6$ is required to generate a ciphertext over a message, nonce and header tuple (m, N, H) using k_{p_i} (or the ephemeral key-wrapped key k' encrypted under k_{p_i}) for p_i, $\mathcal{B}_6$ instead queries (m, N, H) to $\mathcal{C}_{\mathsf{AUTH},i}$'s encryption oracle. Similarly, if $\mathcal{B}_6$ is required to decrypt a ciphertext, nonce, header tuple (C, N, H) for k_{p_i} (or the ephemeral key-wrapped key k' encrypted under k_{p_i}) for parameter set p_i, $\mathcal{B}_6$ instead queries (C, N, H) to $\mathcal{C}_{\mathsf{PRIV},i}$'s decryption oracle. We note that by **Game 1** and the definition of the security experiment, all AEAD keys replaced in this way were already uniformly random and independent of the protocol flow, and $\mathcal{A}$ cannot learn these by issuing a $\mathsf{Corrupt}$ query.

If the bit b sampled by $\mathcal{C}_{\mathsf{PRIV},i}$ is 0, then the $\mathcal{C}_{\mathsf{PRIV},i}$ encrypts the message m honestly, and we are in **Game 1**. Otherwise, if the bit b sampled by the $\mathcal{C}_{\mathsf{PRIV},i}$ is 1, then $\mathcal{C}_{\mathsf{PRIV},i}$ simply samples a ciphertext uniformly at random, and now the ciphertext is uniformly random and completely independent of the protocol flow and we are in **Game 1**. Any $\mathcal{A}$ that could distinguish this change can be directly used to break the PRIV security of the PRIV primitive.

We have at most n_a parameter sets that may be used directly as AEAD keys, and at most n_m key-wrapped AEAD keys, thus $\mathsf{Adv}^{\mathcal{A}}_{G_1}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_2}(\lambda) + (n_a + n_m) \cdot \mathsf{Adv}^{\mathsf{auth}}_{\mathcal{B}_2,\mathsf{AEAD}}(\lambda)$.

Case 3 Analysis: By **Game 1** we replaced the ephemeral key k' used to generate the BCB if p is a key-wrapping algorithm. By **Game 2** we replaced all ciphertexts with uniformly random strings regardless of the challenge bit b. Since the behaviour of the security experiment no longer relies of the challenge bit b the $\mathcal{A}$ has no advantage in guessing and thus we have: $\mathsf{Adv}^{\mathcal{A}}_{G_2}(\lambda) = 0$, and it follows that $\mathsf{Adv}^{\mathsf{FSC}}_{\mathsf{BPSec},\mathcal{A},C_3}(\lambda) \leq n_a \cdot \mathsf{Adv}^{\mathsf{dae}}_{\mathsf{DAE},\mathcal{B}_5}(\lambda) + (n_a + n_m) \cdot \mathsf{Adv}^{\mathsf{PRIV}}_{\mathcal{B}_6,\mathsf{AEAD}}(\lambda)$.

□

We now turn to analysing the SFSC security of the StrongBPSec construction.

Theorem 2 (SFSC Security for StrongBPSec)**.** *The* StrongBPSec *protocol presented in Figure 7 is* SFSC*-secure. That is, for any PPT algorithm $\mathcal{A}$ against the* SFSC *security experiment (described in Definition 3)* $\mathsf{Adv}^{\mathsf{SFSC}}_{\mathsf{StrBPSec},\mathcal{A}}(\lambda)$ *is negligible under the* sufcma *and* dae *security of the* MAC *and* DAE *primitives respectively.*

Proof. We note that in the strong FSC game there are five main ways $\mathcal{A}$ can win. In addition to those described in Theorem 1, the SFSC game introduces two new ways for $\mathcal{A}$ to set $\texttt{win} \leftarrow \texttt{true}$. In particular, if $\mathcal{A}$ has successfully dropped message blocks sent by the security source without being detected when the bundle is processed by the security target, or by forging a read receipt for a given message block. The analysis for the three original cases from the FSC game apply here, and so we focus on the other two cases:

C_1: Adversary $\mathcal{A}$ has successfully dropped blocks $\overrightarrow{\mathtt{M}}_i$ from the source-constructed bundle $\overrightarrow{\mathtt{C}}$ without detection and the bundle payload integrity block $\mathsf{BIB_B}$ verifies correctly;
C_2: Adversary$\mathcal{A}$ has successfully forged a read receipt BRB that verifies correctly.

We proceed via a sequence of games, and bound the difference in the $\mathcal{A}$'s advantage in each game with the underlying cryptographic assumptions. We conclude when the adversary reaches a game where the advantage of that game for that case equals 0, which shows that $\mathcal{A}$ cannot win with non-negligible advantage. We begin by dividing the proof into two separate cases corresponding to each win condition (and denote with $\mathsf{Adv}^{\mathsf{SFSC}}_{\mathsf{StrBPSec},\mathcal{A},C_i}(\lambda)$ the advantage of the adversary in winning the strong integrity game in Case i). It follows that $\mathsf{Adv}^{\mathsf{SFSC}}_{\mathsf{StrBPSec},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{SFSC}}_{\mathsf{StrBPSec},\mathcal{A},C_1}(\lambda) + \ldots + \mathsf{Adv}^{\mathsf{SFSC}}_{\mathsf{StrBPSec},\mathcal{A},C_5}(\lambda)$. We define with $\mathsf{Adv}^{\mathcal{A}}_{G_i}(\lambda)$ the advantage of $\mathcal{A}$ in Game i. We begin with Case 1.

Case 1: Adversary $\mathcal{A}$ wins by causing some party π^i to accept bundle $\overrightarrow{\mathtt{C}}$ from a security source $\pi^{\overrightarrow{\mathtt{C}}_{\mathtt{PB.id,src}}}$ with missing blocks $\overrightarrow{\mathtt{M}}_i$. By the definition of the case, we know that the adversary $\mathcal{A}$ has not been able to Corrupt any keys $(k_p, k_p^{\mathtt{RR}}, k_p^{\mathtt{BB}})$ such that $k_p^{\mathtt{BB}}$ was used to verify (or key-wrap keys to verify) the $\mathsf{BIB_B}$. We proceed via the following series of games.

Game 0 This is the SFSC security game in Case 1: $\mathsf{Adv}^{\mathsf{SFSC}}_{\mathsf{StrBPSec},\mathcal{A},C_1}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_0}(\lambda)$.

Game 1 This game proceeds identically to **Game 1** of **Case 1** of Theorem 1, with the same reductions and bounds. Thus we have $\mathsf{Adv}^{\mathcal{A}}_{G_0}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_1}(\lambda) + n_a \cdot \mathsf{Adv}^{\mathsf{dae}}_{\mathsf{DAE},\mathcal{B}_1}(\lambda)$.

Game 2 In this game, we introduce an abort event that triggers if any un-Corrupted party "accepts" a message $\overrightarrow{\mathtt{M}}$ and no honest party generated a message with the same payload and of the same length. Specifically, this occurs if $\mathcal{A}$ is able to successfully forge $\mathsf{BIB_B}$ using a key associated with a parameter set $\overrightarrow{\mathtt{P}}_i$ such that $\mathsf{Corrupt}(\cdot, \overrightarrow{\mathtt{P}}_i)$ was not issued. This game proceeds identically to **Game 2** of **Case 1** of **Theorem 1**, with the same reductions and bounds. We have at most n_a parameter sets that may be used directly as MAC keys, and at most n_m key-wrapped MAC keys, thus $\mathsf{Adv}^{\mathcal{A}}_{G_1}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_2}(\lambda) + (n_a + n_m) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathcal{B}_2,\mathsf{MAC}}(\lambda)$.

Case 1 Analysis: By definition of **Case 1** we know that $\mathcal{A}$ has caused $\texttt{win} \leftarrow \texttt{true}$ by forcing some party π^i to accept a message with "missing blocks" $\overrightarrow{\texttt{M}}$. Specifically, if π^i accepts a message whilst incorrectly believing that the original bundle had a different amount of blocks than generated. By definition of the security experiment we know that $\mathcal{A}$ has not issued $\mathsf{Corrupt}(\cdot, p)$ such that π^i used $k_p^{\texttt{BB}}$ to verify the $\mathsf{BIB_B}$ block. By **Game 1** we replaced the ephemeral key k' used to generate the $\mathsf{BIB_B}$ if p is a key-wrapping algorithm. By **Game 2** we added an abort query that prevents $\mathcal{A}$ from forging $\mathsf{BIB_B}$ blocks. We note that the $\mathsf{BIB_B}$ is computed over each key identifier that was used by the source node SN and the number of target blocks that the key identifier was used for. Thus, if π^i believed that SN generated an incorrect number of blocks, then the bundle integrity block $\mathsf{BIB_B}$ for $\overrightarrow{\texttt{M}}_{\texttt{PLD}}$ would not verify correctly. Since we abort if the adversary forges their own valid $\mathsf{BIB_B}$, by **Game 2** π^i aborts and thus $\mathcal{A}$ cannot win the game. Thus we have: $\mathsf{Adv}^{\mathcal{A}}_{G_2}(\lambda) = 0$, and it follows that $\mathsf{Adv}^{\mathsf{SFSC}}_{\mathsf{StrBPSec},\mathcal{A},C_1}(\lambda) \leq n_a \cdot \mathsf{Adv}^{\mathsf{dae}}_{\mathsf{DAE},\mathcal{B}_1}(\lambda) + (n_a + n_m) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathcal{B}_2,\mathsf{MAC}}(\lambda)$. Now we transition to **Case 2**.

Case 2: $\mathcal{A}$ wins by causing a party π^i to accept a read receipt BRB under key $k_p^{\texttt{RR}}$ but no read receipt was generated by an honest party.. By the definition of the case, we know that the adversary $\mathcal{A}$ has not issued $\mathsf{Corrupt}(\cdot, p)$. We proceed via the following series of games.

Game 0 This is the SFSC security game in Case 2: $\mathsf{Adv}^{\mathsf{SFSC}}_{\mathsf{StrBPSec},\mathcal{A},C_2}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_0}(\lambda)$.

Game 1 Let n_a be the total number of parameter sets used in the experiment. In this game, for any key-wrapping keys associated with parameter sets that have not been Corrupted, we abort if $\mathcal{A}$ forges a DAE ciphertext, and replace honestly generated key-wrapped keys with uniformly random values. This proceeds identically to the reduction described in **Game 1** of **Case 1** and we find: $\mathsf{Adv}^{\mathcal{A}}_{G_0}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_1}(\lambda) + n_a \cdot \mathsf{Adv}^{\mathsf{dae}}_{\mathsf{DAE},\mathcal{B}_1}(\lambda)$.

Game 2 In this game, we introduce an abort event that triggers if any un-Corrupted party "accepts" a message $\overrightarrow{\texttt{M}}$ with a read receipt BRB (generated using $k_p^{\texttt{RR}}$ and no honest party generated BRB. Specifically, this occurs if $\mathcal{A}$ is able to successfully forge BRB using a key associated with a parameter set $\overrightarrow{\texttt{P}}_i$ such that $\mathsf{Corrupt}(\cdot, \overrightarrow{\texttt{P}}_i)$ was not issued. This game proceeds similarly (up to a change in notation) to **Game 2** of **Case 1** of **Theorem 1**, with the same reductions and bounds. We have at most n_a parameter sets that may be used directly as MAC keys, and at most n_m key-wrapped MAC keys, thus $\mathsf{Adv}^{\mathcal{A}}_{G_1}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{G_2}(\lambda) + (n_a + n_m) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathcal{B}_2,\mathsf{MAC}}(\lambda)$. *Case 2 Analysis:* By **Case 2** we know that a party π^i that verifies a BRB using $k_p^{\texttt{RR}}$ (or a key-wrapped key encrypted under $k_p^{\texttt{R}}$) that sets $\texttt{win} \leftarrow \texttt{true}$ such that $\mathcal{A}$ has not issued $\mathsf{Corrupt}(i, p)$ will abort. Thus $\mathcal{A}$ cannot win the game in **Case 2** and thus we have $\mathsf{Adv}^{\mathcal{A}}_{G_2}(\lambda) = 0$, and it follows that $\mathsf{Adv}^{\mathsf{SFSC}}_{\mathsf{StrBPSec},\mathcal{A},C_2}(\lambda) \leq n_a \cdot \mathsf{Adv}^{\mathsf{dae}}_{\mathsf{DAE},\mathcal{B}_1}(\lambda) + (n_a + n_m) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathcal{B}_2,\mathsf{MAC}}(\lambda)$.

□

6 Conclusion

Space networking follows different restrictions than standard, terrestrial Internet-style networking, which as consequently motivated work in developing and deploying suitable networking protocols over the years. However, the cryptographic security of such, including DTN protocols like BPSec, has largely escaped formal cryptographic analysis, leading to a lack of understanding of how well the security intentions of such protocols are realized. This work provides the first formal cryptographic analysis and provable security treatment of one such protocol and lays a ground work for further analyses in this area. Furthermore, we propose StrongBPSec, a strong alternative to BPSec with additional integrity guarantees. We note that integrating our StrongBPSec with read receipts into existing BPSec instantiations and implementations can be achieved through multiple

approaches: the existing standard could be extended with a new read receipt block type (e.g. as an Other Security Block [BM22, Section 10]); or, more conveniently, read receipts may also be introduced as an extension to the existing BIB block type in the form of a specialized BIB block. We leave exploring the specifics of how StrongBPSec can be incorporated within the existing BPSec paradigm to future work. We highlight, regardless of how it is integrated, the true value of StrongBPSec lies in adopting its read receipts with additional integrity guarantees within the current BPSec and BPSec Default Security Context standards. We further leave finding more efficient *key-wrapping* solutions and reducing the security overhead of BPSec to future work.

References

[ACSA11] Naveed Ahmad, Haitham Cruickshank, Zhili Sun, and Muhammad Asif. Pseudonymised communication in delay tolerant networks. In *2011 Ninth Annual International Conference on Privacy, Security and Trust*, pages 1–6, 2011. doi:10.1109/PST.2011.5971956.

[AKG+07] N. Asokan, Kari Kostiainen, Philip Ginzboorg, Jörg Ott, and Cheng Luo. Applicability of identity-based cryptography for disruption-tolerant networking. In *Proceedings of the 1st International MobiSys Workshop on Mobile Opportunistic Networking*, MobiOpp '07, page 52–56. Association for Computing Machinery, 2007. URL: https://doi-org.libproxy.nps.edu/10.1145/1247694.1247705, doi:10.1145/1247694.1247705.

[BBC17] Fatima Zohra Benhamida, Abdelmadjid Bouabdellah, and Yacine Challal. Using delay tolerant network for the Internet of Things: Opportunities and challenges. In *2017 8th International Conference on Information and Communication Systems (ICICS)*, pages 252–257, 2017. doi:10.1109/IACS.2017.7921980.

[BFB22] Scott Burleigh, Kevin Fall, and Edward J. Birrane. Bundle Protocol Version 7. RFC 9171, January 2022. URL: https://www.rfc-editor.org/info/rfc9171, doi:10.17487/RFC9171.

[BFR08] Scott C. Burleigh, Stephen Farrell, and Manikantan Ramadas. Licklider Transmission Protocol - Specification. RFC 5326, September 2008. URL: https://www.rfc-editor.org/info/rfc5326, doi:10.17487/RFC5326.

[BH17] Colin Boyd and Britta Hale. Secure Channels and Termination: The Last Word on TLS. In *Latincrypt*, 2017. URL: https://api.semanticscholar.org/CorpusID:3648242.

[BHMS16] Colin Boyd, Britta Hale, Stig Frode Mjølsnes, and Douglas Stebila. From Stateless to Stateful: Generic Authentication and Authenticated Encryption Constructions with Application to TLS. In *Proceedings of the RSA Conference on Topics in Cryptology - CT-RSA 2016 - Volume 9610*, page 55–71, 2016. doi:10.1007/978-3-319-29485-8_4.

[Bir23] Edward Birrane. *Securing Delay-Tolerant Networks with BPSec*. Wiley, 2023. URL: https://ieeexplore.ieee.org/servlet/opac?bknumber=10015530.

[BKN02] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in SSH: provably fixing the SSH binary packet protocol. In *Proceedings of the 9th ACM conference on Computer and Communications Security (CCS)*, pages 1–11, 2002. doi:10.1145/586110.586112.

[BM22] Edward J. Birrane and Kenneth McKeever. Bundle Protocol Security (BPSec). RFC 9172, January 2022. URL: https://www.rfc-editor.org/info/rfc9172, doi:10.17487/RFC9172.

[BMM+15] Christian Badertscher, Christian Matt, Ueli Maurer, Phillip Rogaway, and Björn Tackmann. Augmented Secure Channels and the Goal of the TLS 1.3 Record Layer. In *Proceedings of the 9th International Conference on Provable Security - Volume 9451*, ProvSec 2015, page 85–104. Springer-Verlag, 2015. doi:10.1007/978-3-319-26059-4_5.

[BWH22] Edward J. Birrane, Alex White, and Sarah Heiner. Default Security Contexts for Bundle Protocol Security (BPSec). RFC 9173, January 2022. URL: https://www.rfc-editor.org/info/rfc9173, doi:10.17487/RFC9173.

[Dwo04] Morris Dworkin. Request for Review of Key Wrap Algorithms. Cryptology ePrint Archive, Paper 2004/340, 2004. https://eprint.iacr.org/2004/340. URL: https://eprint.iacr.org/2004/340.

[FGJ24] Marc Fischlin, Felix Günther, and Christian Janson. Robust Channels: Handling Unreliable Networks in the Record Layers of QUIC and DTLS 1.3. *J. Cryptol.*, 37(2), jan 2024. doi:10.1007/s00145-023-09489-9.

[FGMP15] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G Paterson. Data is a stream: Security of stream-based channels. In *Advances in Cryptology–CRYPTO 2015, Proceedings, Part II 35*, pages 545–564. Springer, 2015. doi:10.1007/978-3-662-48000-7_27.

[FK11] S. Frankel and S. Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071, February 2011. URL: https://datatracker.ietf.org/doc/html/rfc6071.

[fSDS20a] Consultative Committee for Space Data Systems. CCSDS File Delivery Protocol (CFDP). Technical Report CSDS 727.0-B-5, Consultative Committee for Space Data Systems, 2020. URL: https://public.ccsds.org/Pubs/727x0b5.pdf.

[fSDS20b] Consultative Committee for Space Data Systems. Space Packet Protocol. Technical Report CSDS 133.0-B-2, Consultative Committee for Space Data Systems, 2020. URL: https://public.ccsds.org/Pubs/133x0b2e1.pdf.

[fSDS23] Consultative Committee for Space Data Systems. Overview of Space Communications Protocols. Technical Report CCSDS 130.0-G-4, Consultative Committee for Space Data Systems, Washington DC, USA, April 2023. URL: https://public.ccsds.org/Pubs/130x0g4e1.pdf.

[GM17] Felix Günther and Sogol Mazaheri. A Formal Treatment of Multi-key Channels. In *Advances in Cryptology–CRYPTO 2017 Proceedings, Part III 37*, pages 587–618. Springer, 2017. doi:10.1007/978-3-319-63697-9_20.

[HS02] Russ Housley and Jim Schaad. Advanced Encryption Standard (AES) Key Wrap Algorithm. RFC 3394, October 2002. URL: https://www.rfc-editor.org/info/rfc3394, doi:10.17487/RFC3394.

[KZH07] Aniket Kate, Gregory M. Zaverucha, and Urs Hengartner. Anonymity and security in delay tolerant networks. In *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops - SecureComm 2007*, pages 504–513, 2007. doi:10.1109/SECCOM.2007.4550373.

[LML14] Xixiang Lv, Yi Mu, and Hui Li. Non-Interactive Key Establishment for Bundle Security Protocol of Space DTNs. *IEEE Transactions on Information Forensics and Security*, 9(1):5–13, 2014. doi:10.1109/TIFS.2013.2289993.

[Man23a] Catherine G. Manning. Delay/Disruption Tolerant Networking Overview, 2023. URL: https://www.nasa.gov/technology/space-comms/delay-disruption-tolerant-networking-overview/.

[Man23b] Catherine G. Manning. Frequently Asked Questions, 2023. URL: https://www.nasa.gov/technology/space-comms/delay-disruption-tolerant-networking-faq/.

[MKK17] Sofia Anna Menesidou, Vasilios Katos, and Georgios Kambourakis. Cryptographic Key Management in Delay Tolerant Networks: A Survey. *Future Internet*, 9(3):26, 2017. doi:10.3390/fi9030026.

[Res18] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. Technical report, Internet Engineering Task Force (IETF). RFC 8446, 2018.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, page 98–107. Association for Computing Machinery, 2002. doi:10.1145/586110.586125.

[RS06] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 373–390. Springer, 2006. doi:10.1007/11761679_23.

[RSKW17] Signe Rüsch, Dominik Schürmann, Rüdiger Kapitza, and Lars Wolf. Forward Secure Delay-Tolerant Networking. In *Proceedings of the 12th Workshop on Challenged Networks*, CHANTS '17, page 7–12. Association for Computing Machinery, 2017. doi:10.1145/3124087.3124094.

[SDM04] Paul Syverson, Roger Dingledine, and Nick Mathewson. Tor: The Second Generation Onion Router. In *Usenix Security*, pages 303–320. USENIX Association Berkeley, CA, 2004.

[Sip24] Brian Sipos. DTN Bundle Protocol Security (BPSec) COSE Context. Internet-Draft draft-ietf-dtn-bpsec-cose-04, Internet Engineering Task Force, July 2024. Work in Progress. URL: https://datatracker.ietf.org/doc/draft-ietf-dtn-bpsec-cose/04/.

[TBW+07] Leigh Torgerson, Scott C. Burleigh, Howard Weiss, Adrian J. Hooke, Kevin Fall, Dr. Vinton G. Cerf, Keith Scott, and Robert C. Durst. Delay-Tolerant Networking Architecture. RFC 4838, April 2007. URL: https://www.rfc-editor.org/info/rfc4838, doi:10.17487/RFC4838.

[ZSS+14] Jian Zhou, Meina Song, Junde Song, Xian-Wei Zhou, and Liyan Sun. Autonomic Group Key Management in Deep Space DTN. *Wirel. Pers. Commun.*, 77(1):269–287, July 2014. doi:10.1007/s11277-013-1505-1.

A Cryptographic Assumptions

In this section we describe the cryptographic primitives that are used to build BPSec and define their security. Specifically, we use deterministic authentication encryption DAE

to model the BPSec key wrap algorithm, authenticated encryption with associated data AEAD to model their use of symmetric encryption and message authentication codes MAC to model their use of authentication primitives. We begin by introducing the formalism and security for DAE by repeating the definitions of Rogaway and Shrimpton [RS06] who introduced this primitive.

Definition 5 (Deterministic Authenticated Encryption)**.** A deterministic authenticated encryption scheme DAE is a tuple $\mathsf{DAE} = \{\mathcal{K}, \mathsf{Enc}, \mathsf{Dec}\}$ with associated with key space $\mathcal{K}$, message space $\mathcal{M} \subseteq \{0,1\}^*$ and headers $\mathcal{H} \subseteq \{0,1\}^{**}$. The key space $\mathcal{K}$ is a set of strings or infinite strings endowed with a distribution. For a practical scheme there must be a probabilistic algorithm that samples from $\mathcal{K}$, and we identify this algorithm with the distribution it induces. We denote by $\mathsf{DAE.Enc}_K(H, M)$ the deterministic DAE encryption algorithm that takes as input a key $K \in \mathcal{K}$, a header $H \in \mathcal{H}$ and a message $M \in \mathcal{M}$ and outputs either a ciphertext $C \in \{0,1\}^*$ or a distinguished failure symbol $\perp$. We denote by $\mathsf{DAE.Dec}_K(H, C)$ the deterministic DAE decryption algorithm that takes as input a key $K \in \mathcal{K}$, a header $H \in \mathcal{H}$ and a ciphertext C and returns a string M', which is either in the message space $\mathcal{M}$ or a distinguished failure symbol $\perp$. We assume that $M \in \mathcal{M} \implies \{0,1\}^{|M|} \subseteq \mathcal{M}$. The ciphertext space is $C = \{\mathsf{DAE.Enc}_K(H, M) : K \in \mathcal{K}, H \in \mathcal{H}, M \in \mathcal{M}\}$. Correctness of an DAE scheme requires that $\mathsf{DAE.Dec}_K(H, (\mathsf{DAE.Enc}_K(H, M)) = M$ for all K, H, M in the appropriate space.

We note that in our construction presented in Figure 7 we have adopted the notation KW.Enc and KW.Dec instead of DAE.Enc and DAE.Dec, respectively, as defined above. We wanted our notation to adhere to the vocabulary of the BPSec Default Security Context [BWH22] and thus WLOG we substituted DAE with KW within our construction.

Next, we describe the security of DAE schemes. On a high level a DAE scheme ensures confidentiality of the underlying plaintext and authenticity of the ciphertexts. We provide this in Definition 6.

Definition 6 (DAE Security)**.** Let $\mathsf{DAE} = \{\mathcal{K}, \mathsf{Enc}, \mathsf{Dec}\}$ be a DAE scheme with header space H, message space M, and expansion function e. The advantage of adversary $\mathcal{A}$ in breaking dae-security is defined as

$$\mathsf{Adv}^{\mathsf{dae}}_{\mathsf{DAE},\mathcal{A}}(\lambda) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{Enc}_K(.,.),\mathsf{Dec}_K(.,.)} \implies 1\right] - \Pr\left[\mathcal{A}^{\$(.,.),\perp(.,.)} \implies 1\right].$$

Upon query $H \in \mathcal{H},\ M \in \mathcal{M}$, $\mathcal{A}$'s random oracle $\$(.,.)$ returns a random string of length $|M| + e(H, M)$. The $\perp(.,.)$ oracle returns $\perp$ on every input. We assume that $\mathcal{A}$ does not ask (H, C) of its right oracle if some previous left oracle query (H, M) returned C; does not ask (H, M) of its left oracle if some previous right-oracle query (H, C) returned M; does not ask left queries outside of $H \times M$; and does not repeat a query.

Definition 7 (Message Authentication Code (MAC) security)**.** A message authentication code (MAC) scheme is a tuple of algorithms $\mathsf{MAC} = \{\mathsf{KGen}, \mathsf{Tag}\}$ where:

- KGen is a probabilistic key generation algorithm taking input a security parameter λ and returning a symmetric key k.
- Tag is a potentially probabilistic algorithm that takes as input a symmetric key k and an arbitrary message m from the message space $\mathcal{M}$ and returns a tag τ.

Security is formulated via the following game that is played between a challenger $\mathcal{C}$ and an algorithm $\mathcal{A}$:

1. The challenger samples $k \xleftarrow{\$} \mathcal{K}$
2. The adversary may adaptively query the challenger; for each query value m_i the challenger responds with $\tau_i = \mathsf{Tag}(k, m_i)$

3. The adversary outputs a pair of values (m^*, τ^*). such that $(m^*, \tau^*) \notin \{(m_0, \sigma_0), \ldots (m_i, \sigma_i)\}$

We define the advantage of $\mathcal{A}$ in breaking the strong unforgeability property of a MAC MAC under chosen-message attack to be:

$$\mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{MAC},\mathcal{A}}(\lambda) = \Pr\left((m^*, \tau^*) \notin \{(m_0, \sigma_0), \ldots (m_i, \sigma_i)\}\right)$$

We say that MAC is classically sufcma-secure if, for all PPT algorithms $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{MAC},\mathcal{A}}(\lambda)$ is negligible in the security parameter λ.

Our definition for AEAD-PRIV and AEAD-auth closely follow the work of Rogaway [Rog02] which we describe next for completion.

Definition 8 (AEAD Security)**.** We define an *authenticated-encryption scheme with associated-data (an AEAD-scheme)* as a three-tuple $\Pi = (\mathcal{K}, \mathsf{Enc}, \mathsf{Dec})$. Associated to Π are sets of strings $\mathsf{Nonce} = \{0,1\}^n$ and $\mathsf{Message} \subseteq \{0,1\}^*$, the latter having a linear-time membership test and satisfying $M \in \mathsf{Message} \implies M' \in \mathsf{Message}$ for any M' of the same length as M. Further associated to Π is also a set $\mathsf{Header} \subseteq \{0,1\}^*$ that has a linear-time membership test. The key space $\mathcal{K}$ is a finite nonempty set of strings. The encryption algorithm Enc is a deterministic algorithm that takes strings $K \in \mathcal{K}$ and $N \in \mathsf{Nonce}$ and $H \in \mathsf{Header}$ and $M \in \mathsf{Message}$. It returns a string $\mathcal{C} = \mathsf{Enc}_K^{N,H}(M) = \mathsf{Enc}_K(N, H, M)$. Decryption algorithm Dec is a deterministic algorithm that takes strings $K \in \mathcal{K}$ and $N \in \mathsf{Nonce}$ and $H \in \mathsf{Header}$ and $\mathcal{C} \in \{0,1\}^*$. It returns $\mathsf{Dec}_K^{N,H}(\mathcal{C})$, which is either a string in $\mathsf{Message}$ or the distinguished failure symbol $\perp$. The correctness of the AEAD scheme requires that $\mathsf{Dec}_K^{N,H}(\mathsf{Enc}_K^{N,H}(M)) = M$ for all $K \in \mathcal{K}$ and $N \in \mathsf{Nonce}$ and $H \in \mathsf{Header}$ and $M \in \mathsf{Message}$. Some linear-time computable length function ℓ is given by $|\mathsf{Enc}_K^N(M)| = \ell(|M|)$.

AEAD PRIV SECURITY: Let $\Pi = (\mathcal{K}, \mathsf{Enc}, \mathsf{Dec})$ be AEAD-scheme with length function ℓ. Let $\$(\cdot,\cdot,\cdot)$ be an oracle that, upon input returns (N, H, M), returns a random string of $\ell(|M|)$ bits. The advantage of adversary $\mathcal{A}$ in breaking AEAD PRIV-security is defined as

$$\mathsf{Adv}^{\mathsf{PRIV}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{Enc}_K(\cdot,,,\cdot)} \implies 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,,,\cdot)} \implies 1\right].$$

AEAD AUTH SECURITY: Let $\Pi = (\mathcal{K}, \mathsf{Enc}, \mathsf{Dec})$ be AEAD-scheme, and let $\mathcal{A}$ be an adversary with access to an orcale $\mathsf{Enc}_K(\cdot,\cdot,\cdot)$. We say that $\mathcal{A}$ *forges* (for this key K and on some particular run) if $\mathcal{A}$ outputs $(N, H, \mathcal{C})$ where $\mathsf{Dec}_K^{N,H}(\mathcal{C}) \neq \perp$ and $\mathcal{A}$ did not query $\mathsf{Enc}_K^{N,H}(M)$ to the encryption oracle. We say that an AEAD scheme AEAD is auth- secure if for all PPT algorithms $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)$ is negligible in the security parameter λ.