BERMUDA: A BPSec-Compatible Key Management Scheme for DTNs

Fiona Fuchs[1,2], Felix Walter[1], and Florian Tschorsch[2]

[1] D3TN GmbH, Poststr. 37, 01159 Dresden, Germany
fiona.fuchs@d3tn.com, felix.walter@d3tn.com
[2] Dresden University of Technology, 01062 Dresden, Germany
fiona.fuchs@mailbox.tu-dresden.de, florian.tschorsch@tu-dresden.de

Abstract. Delay- and Disruption-tolerant Networks (DTNs) enable communication in challenging environments like space and underwater. Despite the need for secure communication, key management remains an unresolved challenge in DTNs. Both DTN security protocols, BSP and BPSec, explicitly exclude key management from their scope, and research in this area remains limited. Traditional Internet-based key management methods are largely unsuitable due to the unique constraints of DTNs. In this paper, we present BERMUDA, a BPSec-compatible key management framework for unicast messaging. Our approach combines established building blocks, including a hierarchical PKI and ECDH, with an adapted version of NOVOMODO for certificate revocation. To evaluate its applicability, we implement a DTN chat application as an example use case and analyze the system's scalability. While our findings demonstrate the feasibility of BERMUDA for DTNs, we also show limitations related to scalability and computational load in resource-constrained scenarios. By bridging the gap between conceptual designs and practical deployment, this work advances key management research in DTNs, contributing to secure communication in these demanding networks.

Keywords: Delay- and Disruption-tolerant Networks (DTNs) · Key Management · BPSec

1 Introduction

Delay- and Disruption-tolerant Networks (DTNs) enable communication in challenging environments marked by frequent disruptions, prolonged propagation delays, intermittent connectivity, and high error rates [12]. Conventional networking protocols, such as those based on TCP/IP, fail to operate effectively under such conditions. DTNs address these limitations by introducing a novel networking architecture based on a store-and-forward approach, where data is transmitted when conditions allow and stored at intermediate nodes until a viable route to the destination emerges [12]. This approach is implemented through the Bundle Protocol (RFC 5050, RFC 9171)—an overlay protocol tailored for DTNs. DTNs are applicable in diverse scenarios, such as sensor-based, underwater acoustic, vehicular ad hoc, satellite and deep-space, as well as terrestrial wireless networks where sustained end-to-end connectivity is often infeasible [12].

This paper has been accepted for publication at the IFIP SEC International Workshop on Network and Distributed Systems Security (WNDSS 2025) and will be available at Springer.

Despite their innovative architecture, DTNs face significant challenges in securing communications. The Bundle Protocol Security (BPSec, RFC 9172) represents the standardized security framework for DTNs, but—like its predecessor BSP (RFC 6257)—explicitly excludes key management from its scope. Conventional key management approaches developed for the Internet are widely deemed unsuitable for DTNs due to the unique constraints [5, 13], such as intermittent connectivity and long delays. This gap highlights the need for research into key management schemes compatible with BPSec and tailored to DTN challenges.

In this paper, we present BERMUDA, a **B**PSec-Compatible Key **E**stablishment and **R**evocation **M**echanism for **U**nicast Messaging in **D**elay-tolerant **A**pplications, as a suitable key management solution for DTNs. To the best of our knowledge, this is the first approach specifically designed for BPSec compatibility, allowing seamless integration into BPSec-secured communication, whereas previous solutions were based on BSP. Based on an analysis of key management requirements, we design, implement, and evaluate our solution. We develop a DTN chat application as a use case to illustrate our research, suitable for scenarios where remote teams in resource-constrained environments require reliable communication.

By bridging the gap between conceptual design and practical deployment, we seek to contribute to the broader field of Delay- and Disruption-tolerant Networking, advancing the understanding of key management within this domain. Our main contributions are as follows: (1) We propose BERMUDA, a novel BPSec-compatible key management scheme for DTNs that notably supports key revocation in unicast messaging scenarios; and (2) we implement, analyze, and evaluate BERMUDA, demonstrating its feasibility for DTN applications.

The paper is structured as follows: Section 2 defines the system model, threat analysis, and key management requirements. Section 3 introduces BERMUDA. Section 4 analyzes it in the DTN context and empirically evaluates its scalability. Related work is reviewed in Section 5, followed by a conclusion in Section 6.

2 System Requirements

System Model This work focuses on bidirectional point-to-point communication in a DTN context, limited to interplanetary communication within our solar system and terrestrial communication. We assume all messages exchanged between nodes are secured using BPSec (RFC 9172), which provides mechanisms to ensure the confidentiality and integrity of data in DTNs. Specifically, all messages are encapsulated in a Bundle Confidentiality Block (BCB), which encrypts the content of the message to protect it from unauthorized access.

Our proposed approach assumes a maximum one-way end-to-end transmission time of 24 hours, based on the longest possible communication distance in our model: when Neptune and Uranus are at their aphelia on opposite sides of the Sun, about 7.56 billion kilometers apart. At light speed, such a transmission would take about 7 hours, but additional factors like intermittent connectiv-

ity, processing delays, and planetary or solar obstructions justify 24 hours as a realistic upper bound.

Attacker Model We consider an external attacker aiming to compromise the key management process, while internal threats, such as insider abuse or compromised nodes, are excluded from this paper's scope. The attacker's capabilities include intercepting, modifying, and injecting messages. However, Denial of Service (DoS) attacks and message deletion are beyond the scope of this work and are left for future research. DoS mitigation in DTNs remains a broader research challenge that requires additional measures. Moreover, cryptographic primitives, including encryption algorithms and hash functions, are assumed to be correctly implemented and resistant to current computational attack capabilities, with securely generated keys. If a trusted third party is involved, it is assumed to be trustworthy and secure from compromise, employing authentication to verify the identities of network participants.

Key Management Requirements The key management approach must address the unique challenges of our DTN system model by meeting several requirements. First, it should minimize round trips for key establishment, ideally completing the initial setup in a single round trip and avoiding additional exchanges for subsequent messages. Second, the scheme must be deployable on low-powered devices, such as onboard or IoT systems, by utilizing lightweight cryptographic operations. Third, it should leverage a central trust anchor, but at the same time operate independently of immediate responses from a central entity, with request-response communication limited to long-lived data like certificates provided by a Certificate Authority (CA).

Additionally, the scheme must ensure confidentiality of exchanged keys through strong encryption, verify their integrity, and provide authentication mechanisms for secure key establishment between previously untrusting parties. It should also include a mechanism for timely key revocation to prevent misuse of compromised or outdated keys. Finally, the approach must be compatible with and usable alongside the default security contexts of BPSec, as defined in RFC 9173.

In this work, we assume that provisioning a single central authority is acceptable for the given system. However, we design the approach in a way that allows for using multiple authorities to ensure the overall system's continuous availability.

3 BERMUDA Key Management

BERMUDA builds on established cryptographic primitives, such as a hierarchical Public Key Infrastructure (PKI) and Elliptic Curve Diffie-Hellman (ECDH) for key exchange, which are used without significant modifications. The core innovation lies in extending the NOVOMODO design [7] to manage the certificate lifecycle in a DTN context, by incorporating a stapling mechanism tailored to the requirements.

3.1 Basic System Components

Public Key Infrastructure A hierarchical PKI is the primary framework for key management on the Internet [2]. It employs digital certificates to authenticate and bind public keys to their respective owners. A trusted Certificate Authority (CA) issues certificates signed with its root private key after verifying the requester's identity and ensuring secure private key storage.

We opt for a PKI-based approach to address authentication and trust in DTNs, building on its established security properties in Internet applications. Although RFC 9173 recommends symmetric-key cryptography for BPSec due to the challenges of deploying asymmetric-key infrastructures with central authorities in DTNs, we argue that PKI remains viable for many DTN use cases and offers a more efficient and robust solution. Symmetric cryptography is less computationally demanding but often incurs higher communication overhead for secure key establishment [3], which asymmetric cryptography helps avoid.

While we assume a single CA in this paper, the approach also supports multiple CAs—e.g., one per planet—to reduce single points of failure and end-to-end delays. The certificates in our system follow the structure proposed in NOVOMODO [7], consisting of: a unique serial number (SN), the public key (PK) generated by the holder, the user identity (U) to identify the holder, issuance (D_1) and expiration (D_2) dates for defining the validity period, a revocation target (Y_1) to confirm the certificate's revocation, and a validity target (X_{365}) to confirm validity if the certificate remains active. Users request certificates from the CA upon joining the system, or when previous ones expire or are revoked.

ECDH Key Exchange Our framework uses ECDH, specifically X25519, for key exchange. The choice of Elliptic Curve Cryptography (ECC) is motivated by its efficiency, offering security levels comparable to RSA with smaller keys and faster computations. ECDH establishes a shared secret over an insecure channel without transmitting the key itself. As a well-established component of secure protocols like TLS 1.3, ECDH represents a reliable solution.

In BERMUDA, the ECDH secret is processed via a key derivation function and used for subsequent AES-GCM encryption, following BPSec's *BCB-AES-GCM* security context (RFC 9173). Performing a new ECDH exchange for every message is impractical in DTNs due to resource constraints and round-trip minimization. Instead, this research assumes that key exchanges occur only when initiating communication with a new partner or when certificates are revoked or expire. The derived ECDH shared secret can also be used for HMAC authentication, aligning with BPSec's *BIB-HMAC-SHA2* security context (RFC 9173).

3.2 Certificate Revocation with BERMUDA

BERMUDA's certificate revocation mechanism builds on NOVOMODO [7], where the revocation status—indicating whether a certificate is valid or revoked—is explicitly requested, similar to OCSP (RFC 6960).

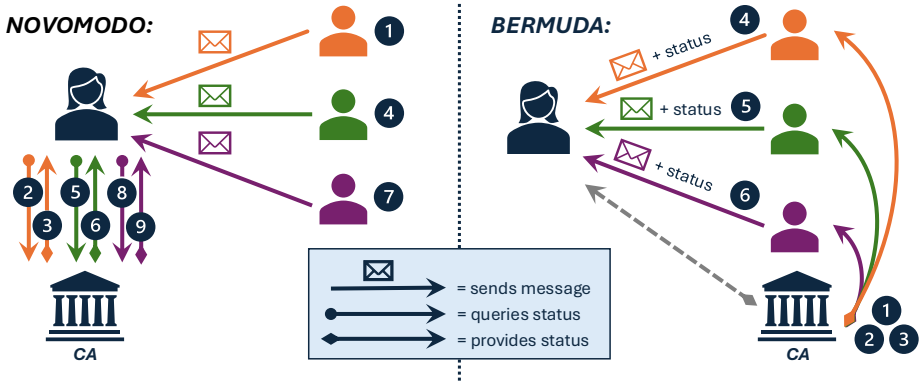

Fig. 1: Comparison of revocation status updates in NOVOMODO and BERMUDA.

However, unlike OCSP, which uses digitally signed CA responses, NOVOMODO achieves this by leveraging hash values that only the CA can compute. These hash values implicitly encode the revocation status, eliminating the need for signatures. As hash computations demand less bandwidth and computational power than digital signatures, NOVOMODO is well-suited for the resource-constrained DTN context, making it the preferred basis for certificate revocation in our BERMUDA approach.

Bhutta et al. [2] previously proposed NOVOMODO for DTNs in combination with CRLs (RFC 5280), but distributing large CRLs reduces routing efficiency and adds overhead. To address this, we adopt NOVOMODO's original design, adapted to meet the requirements of our use case (cf. Section 2). The following sections provide an overview of NOVOMODO and our extensions to its revocation mechanism. The novel mechanism implemented in BERMUDA is shown in Fig. 1.

NOVOMODO During certificate issuance, the CA generates two secret random values, X_0 and Y_0, known only to the CA itself.

Using X_0, the CA generates a hash chain matching the certificate's validity period—set to 365 days in our implementation, as we consider it suitable for various DTN use cases and consistent with the example provided in NOVOMODO [7]. For hashing, we use SHA-256 instead of SHA-1 from the original approach. The final value in the hash chain, X_{365} in our case, serves as the *validity target* and is included in the certificate. The hash chain is computed as $X_i = H^i(X_0)$ for $1 \leq i \leq 365$, where H^i denotes i iterative applications of the hash function (e.g., $X_2 = H(H(X_0) = H(X_1))$.

Using Y_0, the CA computes Y_1 by hashing it once, i.e., $Y_1 = H(Y_0)$. This value is included in the certificate as the *revocation target*. After generating the certificate, the CA stores the user's information, notably the secret seeds X_0 and Y_0.

When the revocation status of a user's certificate is requested, the CA either answers with X_{365-i} or Y_0, depending on the certificate's validity:

1. **Certificate is still valid:** Here, the CA calculates the number of days i since the certificate was issued and computes X_{365-i}, assuming a 365-day validity period with daily status checks. For less frequent checks, the calculation of i must be adjusted accordingly. Since X_0—the initial seed of the hash chain—is only known to the CA, and hash functions are one-way, the provided status value can be trusted. The user verifies this value by hashing X_{365-i} exactly i times to confirm it matches X_{365}, which is specified in the certificate. This mechanism also helps mitigate replay attacks, as an attacker cannot reuse a previously issued status value to falsely indicate that a revoked certificate is still valid.
2. **Certificate is revoked:** In this scenario, the CA provides Y_0 to the user. As only Y_1—which is specified in the certificate—is public, only the CA has access to Y_0. The user confirms the revocation status by verifying that $H(Y_0) = Y_1$.

For enhanced efficiency, the CA may optionally store (cache) parts or all of the hash chain (X_0 to X_{365}) instead of only the initial seed (X_0). This approach allows faster lookups by avoiding the need to recompute many hash values on demand, which is especially beneficial in environments with constrained computational resources. However, this optimization increases storage requirements.

Stapling Using NOVOMODO in its original form would require each user to individually request the revocation status for every sender whose messages they receive in a day from the CA. For instance, if Alice receives messages from Bob, Carol, and Dave on a given day Z, she would need to request the revocation statuses of three certificates. This approach places a significant workload on the CA and leads to numerous round trips—both problematic in a DTN context.

To address this, we introduce a stapling mechanism: Instead of requiring receivers to query the CA for each sender's status, the responsibility shifts to the senders. In the example, Bob, Carol, and Dave would each request their own status from the CA and append them to the messages they send to Alice. Alice can then verify the appended statuses directly without querying the CA herself. In BERMUDA, each user appends their value X_{365-i} (for that day) to every message they send, as well as during the initiation of an ECDH key exchange.

For practical deployment in a BPSec implementation, the revocation status can be included in a Bundle Extension Block. To reduce overhead, the status could be appended only to the first message sent to each receiver on a given day. However, this creates a potential DoS scenario if that message is lost or maliciously deleted.

Publish and Subscribe Implementing NOVOMODO with stapling still relies on timely responses from the CA for status requests—now triggered by the sender of messages. To overcome this issue, we propose a revised approach

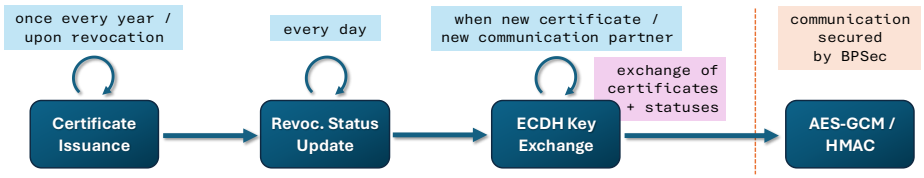

Fig. 2: Key management process in BERMUDA.

where, upon entering the system and requesting a certificate, each user automatically subscribes to (daily) revocation status updates from the CA. Such publish-subscribe models are well-established in DTNs [1]. Each user receives their respective status the day before, ensuring smooth operation on the day it is actually required.

While this approach introduces some overhead and redundancy—since not all users send messages daily and therefore may not require their revocation status every day—it effectively addresses the reliance on timely responses from the CA and minimizes round trips, making it a practical trade-off in the DTN context.

Grace Period To handle potential delays in receiving status updates caused by intermittent connectivity in DTNs, we introduce a grace period. This period defines how many days an outdated revocation status remains valid for communication partners. In this work, we set the grace period to one day, meaning the previous day's status is still acceptable. This value is based on our assumption of a maximum expected end-to-end delay of 24 hours (cf. Section 2). A one-day grace period in BERMUDA ensures that, even if today's status is delayed, including the previous day's status in the message allows communication to proceed without waiting for the updated status to arrive, thereby maintaining reliable operation.

3.3 Summary of the Key Management Process in BERMUDA

Bringing all components together, we outline how they interact in a coherent process (cf. Fig. 2). Upon joining the network, Alice and Bob request a certificate from the CA, which is digitally signed with Ed25519. This process repeats upon certificate expiration or revocation. Additionally, both users automatically subscribe to daily revocation status updates from the CA, transmitted as SHA-256 hash values. These updates are verified upon receipt to ensure integrity and are included with every message to confirm certificate validity.

When Alice initiates communication with Bob, they perform an ECDH key exchange using X25519 by sharing their certificates, which include their public keys and revocation statuses. Each user verifies the other's credentials for authenticity. A symmetric key is then derived from the computed shared secret. This key enables encrypted communication via AES-GCM for confidentiality

Table 1: Message Types Used for Key Management in the DTN Chat Application

Type	Purpose
`REQ_CERT`	Sent by a user to request a certificate from the CA
`ACK_CERT`	CA responds to `REQ_CERT` with a signed certificate for the user
`REQ_ECDH`	Initiates contact by sending the certificate and revocation status
`ACK_ECDH`	Replies to `REQ_ECDH` with partner's certificate and revocation status
`REV_STATUS`	Daily update from the CA with hash of the revocation status
`REQ_CERT_REVOC`	Sent by a user to request the CA to revoke their own certificate

and integrity, or HMAC for integrity alone, as specified in BPSec. A new ECDH key exchange occurs only when a certificate expires, is revoked, or when communicating with a new partner.

4 Evaluation and Discussion

To assess whether BERMUDA performs effectively under the challenging conditions of DTNs, it is analyzed and evaluated against four key metrics: delay tolerance, data overhead, reliability, and scalability.

4.1 Implementation of a DTN Chat Application

To support the evaluation of BERMUDA, we implemented a DTN chat application with AES-GCM encryption using *µD3TN*[3], which handles bundle transmission and network contact configuration. Python was chosen for its extensive library support and compatibility with µD3TN's Python-based tools. For cryptographic algorithms, we used the Python library *Cryptography*[4] due to its broad algorithm support.

In Table 1, we provide an overview of the message types used in the chat application for the key management process. Upon starting the chat application, the pre-initiated users Alice, Bob, and Carol automatically request a certificate from the CA, whose public key is known to all nodes in the system. Identity verification of new users entering the system is considered beyond the scope of our concept. For further details on the implementation, readers are referred to the public repository,[5] which contains the source code and documentation for the DTN chat application, including its cryptographic components, usage examples, and the evaluation setup.

4.2 Analysis of BERMUDA's Suitability for DTNs

Delay Tolerance While BERMUDA minimizes reliance on timely responses, some delay-sensitive processes remain unavoidable. Specifically, certificate is-

[3] https://gitlab.com/d3tn/ud3tn
[4] https://cryptography.io/
[5] https://github.com/raniya02/dtn_crypto_chat

Table 2: Message Component Sizes Assumed for Analysis

Component	Details	Size (Bytes)
Public Key	X25519 key exchange algorithm	32
Digital Signature	Ed25519 signature algorithm	64
NOVOMODO Hashes	SHA-256 hashing algorithm	32
Authentication Tag	AES-GCM, as mandated in RFC 9173	16
Initialization Vector	AES-GCM, as mandated in RFC 9173	12
Serial Number	According to RFC 5280 variable up to 20 bytes for X.509 certificates; assumed for smaller size of DTNs	10
User Name	Based on average name length [6]	14
Issuance/Expiration Date	ISO 8601 format "YYYY-MM-DD"	10

suance relies on a request-response model and is required for subsequent key establishment and message exchange. Similarly, communication cannot begin until key establishment is complete, which requires the delivery of both `REQ_ECDH` and `ACK_ECDH` messages. However, certificate issuance and ECDH key exchanges occur infrequently.

In contrast, regular message exchanges between two users are no more affected by delays than unsecured DTN communications. Additionally, BERMUDA's revocation handling minimizes delay sensitivity by eliminating the need for status requests. The grace period further accommodates potential delays in receiving updates, preventing users from being misclassified as unauthorized. The introduced parameters can be adjusted for especially highly delayed environments: Longer certificate validity reduces the frequency of certificate requests and ECDH exchanges, while extended grace periods account for greater delays in status updates.

Data Overhead in Messages In our application, messages are divided into regular chat messages and key management messages. In Table 2 and 3, we summarize the total sizes of message types and their components. We classify all key management messages used by BERMUDA as overhead, as they are absent in systems without key management. Regular chat messages with BERMUDA incur an additional 32 bytes per message for the added revocation status—compared to BPSec-only messages without key management, which include only the authentication tag and AES-GCM initialization vector apart from the message content itself.

To minimize overhead, BERMUDA employs ECC (X25519, Ed25519) over RSA to reduce key sizes and 32-byte hashes over OCSP's 64-byte signatures, avoiding CRL-based methods that require distributing large datasets. While certificates contribute most to the overhead, extending certificate lifetimes can reduce the frequency of key management messages. For lower security needs, sending revocation updates only once per day eliminates the extra 32 bytes in

Table 3: Approximate Size of a NOVOMODO Certificate and Different Message Types

Type	Content Breakdown (Bytes)	Size (Bytes)
Certificate	Serial No.: 10, User Name: 14, Public Key: 32, Issuance Date: 10, Expiration Date: 10, Validity Hash: 32, Revocation Hash: 32	140
`REQ_CERT`	User Name: 14, Public Key: 32	46
`ACK_CERT`	Certificate: 140, Signature: 64	204
`REQ/ACK_ECDH`	Certificate: 140, Signature: 64, Revocation Status: 32	236
`REV_STATUS`	Revocation Status: 32	32
Message Header	Revocation Status: 32, Tag: 16, IV: 12	60

subsequent messages, matching BPSec-only messaging. However, this increases risks from message loss or tampering.

Reliability BERMUDA employs six distinct message types, introduced in addition to the regular messages exchanged by the users, as listed in Table 1. Each message fulfills a specific role within the key management process. Consequently, the loss of any of these messages would prevent the system from functioning as intended.

The loss of a `REQ_CERT` or `ACK_CERT` message prevents certificate issuance, preventing the user from participating in secure communications. Similarly, losing a `REQ_ECDH` or `ACK_ECDH` message disrupts the key exchange, making secure communication setup impossible. The impact of losing a `REV_STATUS` message depends on the certificate's state: if still valid, the grace period minimizes disruption by allowing for delays until a new update is expected the following day; if revoked, users may unknowingly use a compromised certificate until the grace period is exceeded. Finally, loss of a `REQ_CERT_REVOC` message prevents certificate revocation, potentially leaving compromised certificates valid.

BERMUDA attempts to improve reliability by including the current revocation status with every regular message, rather than limiting it to the first message of the day. However, the system remains heavily reliant on the successful delivery of key management messages. While this dependence is challenging to eliminate, future work could explore mechanisms like time-outs or reliable convergence-layer protocols (such as TCP and LTP) to improve message delivery in error-prone environments. Another option is custody transfer, previously available in version 6 of the Bundle Protocol but removed in version 7, where intermediate nodes remain responsible for retransmission of a bundle until its receipt is confirmed by the recipient.

4.3 Scalability Evaluation

The scalability evaluation examines the system's performance under increasing user loads, focusing on three scenarios: certificate requests, revocation status updates, and ECDH key exchanges.

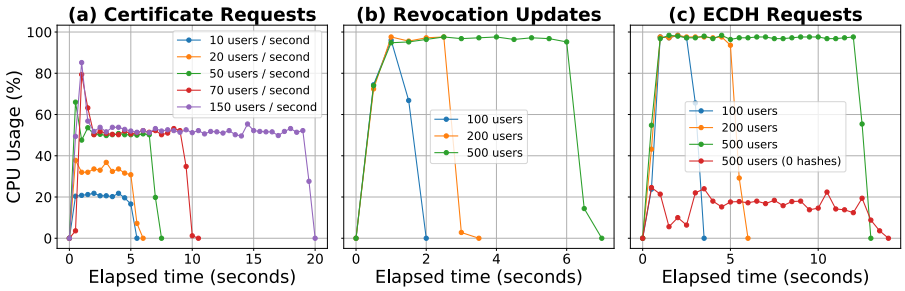

Fig. 3: CPU usage during each of the three scenarios.

Certificate Requests The evaluation measured CPU and memory usage of the CA while processing certificate requests, sampling in a one-second interval, averaged over five repetitions. Each request requires the CA to compute 366 values (365 for validity and one for revocation).

The CA ran on a Raspberry Pi 3 Model B to emulate a resource-constrained environment—ensuring it would also run on higher-powered devices and isolating it from other processes that might affect the evaluation results. User instances ran on a consumer-grade laptop (Intel i5-1235U, 16 GiB RAM). The tests involved 10 to 150 users starting per second over 10 iterations, totaling 100 to 1500 users per run.

In Fig. 3a, we show the measured CPU usage, starting when the process is no longer idle ($> 0\,\%$) and ending when usage returns to $0\,\%$, marking the completion of request processing. The results indicate that CPU load exhibits an initial spike caused by a burst of incoming requests, before stabilizing at approximately $60\,\%$ for higher user loads. Processing duration increased with user count, reflecting the CA's ability to handle larger loads by extending processing time. Notably, even on commodity hardware running a Python implementation of the approach, processing requests for up to 1,500 users took less than 20 seconds. This demonstrates scalability and suitability even for larger deployments. The memory usage of the CA, on the other hand, remained steady at around 25 MiB, indicating minimal strain on resources beyond CPU power. The initial CPU spike is likely attributed to the limited capability of the laptop used to initialize a large number of user instances, suggesting that future evaluations could benefit from employing higher-powered devices for this task.

Revocation Status Updates This scenario evaluated the CA's performance in computing and sending revocation status updates, with CPU and memory usage measured at 0.5-second intervals for finer granularity compared to the certificate request scenario. Each update involved 364 hash calculations per user, representing the worst-case scenario in terms of computational load.

In Fig. 3b, we show the CA's CPU usage over time. The results reveal that all tests exhibited an initial spike in CPU usage, reaching approximately $100\,\%$

as the CA began processing. For larger user loads, this was followed by a short period of high usage—ranging from about 2 s for 100 users to 7 s for 500 users—which suggests potential for further scaling the number of users, given that updates occur only once per day. Again, memory usage was around 25 MiB, highlighting that the primary resource demand stems from CPU-bound processing. To enhance efficiency under heavy loads, future efforts could focus on parallelizing hash calculations.

ECDH Key Exchange Requests This test evaluated the CPU usage of a regular user (referred to as Alice for simplicity) during ECDH key exchange requests, focusing on a worst-case scenario requiring 727 hash calculations per request: 363 for the actual number of days passed since certificate issuance, and 364 for the one-day delay allowed by the grace period, tested only if the 363-day hash is found to be invalid. A comparison with the best-case scenario, where no hash calculations are required (hash chain length of zero), was also included to assess the impact of hashing on CPU usage. The CA remained on the Raspberry Pi but was not measured in this test, with Alice's instance running on a separate Raspberry Pi.

In Fig. 3c, we show Alice's average CPU usage under different conditions. CPU usage peaked at nearly 100 % across all user loads, with processing times increasing from about 4 s (100 users) to 13 s (500 users). Memory usage remained consistently low at approximately 25 MiB. Alice's high CPU usage reflects full utilization of available resources, which is beneficial without competing workloads. While optimizations could further reduce runtime, the tested values already exceed realistic expectations. Unlike the CA, which handles system-wide operations, Alice would typically interact with far fewer users, and key exchanges are infrequent in our model. In practice, she would rarely experience such high request loads, making high scalability less critical. In contrast, the no-hash scenario showed significantly lower but fluctuating CPU usage, which likely stems from the test setup. Without hash calculations, Alice processes ECDH requests faster while her peers have to complete preparatory steps and transmit the data through the network, causing short idle phases where Alice has no active tasks. Further testing with more balanced setups is needed to fully understand the impact of different hash chain lengths.

5 Related Work

Traditional key management protocols such as TLS (RFC 8446), IKE (RFC 2409/7296), and Kerberos [8] face limitations in DTNs due to their reliance on synchronous interactions and multiple round trips. Asynchronous alternatives like PGP (RFC 9580) and the Signal Protocol[6] also fall short, requiring manual verification by participants in the network or assuming responsive central servers.

[6] https://signal.org/docs/

Certificate revocation in conventional PKIs relies on CRLs (RFC 5280) and OCSP (RFC 6960), both problematic in DTNs [2] due to OCSP's reliance on real-time answers from a central server and the high bandwidth demands associated with distributing large CRLs. OCSP stapling (RFC 6961) offers a more suitable compromise, inspiring the stapling mechanism in BERMUDA.

In addition, there are approaches specifically designed for DTNs. Templin and Burleigh [11] define requirements for DTN key management, which Burleigh et al. [4] build upon by presenting a decentralized architecture that regularly distributes certificates, identities, signatures, and revocation data to all nodes. While BERMUDA assumes a CA, it could be extended with a multi-signature scheme. To improve scalability and reduce overhead, we avoid proactively distributing all participant information network-wide.

Many existing DTN proposals reject PKI, assuming a globally available CA is unrealistic [5, 9, 13]. Nonetheless, Bhutta et al. [2] propose a PKI mechanism combining CRLs with hash tables and NOVOMODO. Yet, CRLs remain a bandwidth burden, even when optimized—such as in [14]. Patra et al. [9] propose Identity-Based Cryptography (IBC), where public identifiers act as public keys and a central Private Key Generator (PKG) issues the corresponding private keys. However, IBC relies on frequent PKG responses due to short key lifetimes (leveraging revocation by expiration) and introduces security risks as private keys are managed by a third party. Other DTN approaches assume a secure secondary channel [5] or pre-shared keys [13].

The IETF draft [10] extends the ACME protocol to support DTN node identifier validation. While identity verification lies beyond this work's scope, such efforts could complement BERMUDA in a full DTN security architecture.

6 Conclusion

In this work, we presented BERMUDA, a novel key management scheme for DTNs. Unlike previous proposals, our approach is specifically tailored to BPSec—the current DTN security standard—enabling exchanged keys to support subsequent AES-GCM encryption and HMAC authentication. We leverage well-established PKI techniques, which are typically dismissed by DTN key management research, and adapt them to address end-to-end delays effectively.

Our analysis confirms BERMUDA's feasibility for DTNs, demonstrating good scalability in managing certificate and ECDH requests, as well as efficiently handling revocation status updates. With short processing times, the approach proves to be suitable for small to medium-sized DTN deployments, even on low-power hardware like a Raspberry Pi. Optimizations such as parallelized processing, improved test setups, and lower-level programming languages could further support deployment in large-scale DTNs. The introduced parameters can be adjusted based on expected network conditions and desired security requirements, making BERMUDA a flexible solution for diverse DTN applications.

References

1. Abe, R., Nakamura, Y., Takahashi, O.: A dtn routing scheme based on publish/subscribe model. International Workshop on Informatics (IWIN2013) pp. 23–29 (9 2013)
2. Bhutta, M.N.M., Cruickshank, H., Sun, Z.: Public-key infrastructure validation and revocation mechanism suitable for delay/disruption tolerant networks. IET Information Security **11**(1), 16–22 (2017). https://doi.org/10.1049/iet-ifs.2015.0438
3. Bhutta, N., Ansa, G., Johnson, E., Ahmad, N., Alsiyabi, M., Cruickshank, H.: Security analysis for delay/disruption tolerant satellite and sensor networks. In: 2009 International Workshop on Satellite and Space Communications. pp. 385–389 (2009). https://doi.org/10.1109/IWSSC.2009.5286339
4. Burleigh, S.C., Horres, D., Viswanathan, K., Benson, M.W., Templin, F.: Architecture for Delay-Tolerant Key Administration. Internet-draft, Internet Engineering Task Force (Aug 2018), https://datatracker.ietf.org/doc/draft-burleigh-dtnwg-dtka/02/
5. Lv, X., Mu, Y., Li, H.: Non-interactive key establishment for bundle security protocol of space dtns. IEEE Transactions on Information Forensics and Security **9**(1), 5–13 (2014). https://doi.org/10.1109/TIFS.2013.2289993
6. Maaroufi, M., Landais, P., Messiaen, C., Choquet, R.: Federating patients identities: the case of rare diseases. Orphanet Journal of Rare Diseases **13** (11 2018). https://doi.org/10.1186/s13023-018-0948-6
7. Micali, S.: Novomodo: Scalable certificate validation and simplified pki management. In: 1st Annual PKI Research Workshop Proceedings. pp. 15–25 (Nov 2003), https://www.nist.gov/publications/1st-annual-pki-research-workshop-proceedings
8. Miller, S.P., Neuman, B.C., Schiller, J.I., Saltzer, J.H.: Project Athena: Kerberos Authentication and Authorization System (1988), https://web.mit.edu/saltzer/www/publications/athenaplan/e.2.1.pdf
9. Patra, R., Surana, S., Nedevschi, S.: Hierarchical identity based cryptography for end-to-end security in dtns. In: 2008 4th International Conference on Intelligent Computer Communication and Processing. pp. 223–230 (2008). https://doi.org/10.1109/ICCP.2008.4648376
10. Sipos, B.: Automated Certificate Management Environment (ACME) Delay-Tolerant Networking (DTN) Node ID Validation Extension. Internet-draft, Internet Engineering Task Force (Nov 2024), https://datatracker.ietf.org/doc/draft-ietf-acme-dtnnodeid/16/
11. Templin, F., Burleigh, S.C.: DTN Security Key Management - Requirements and Design. Internet-draft, Internet Engineering Task Force (Apr 2016), https://datatracker.ietf.org/doc/draft-templin-dtn-dtnskmreq/00/
12. Torgerson, L., Burleigh, S.C., Weiss, H., Hooke, A.J., Fall, K., Cerf, V.G., Scott, K., Durst, R.C.: Delay-Tolerant Networking Architecture. RFC 4838 (Apr 2007). https://doi.org/10.17487/RFC4838
13. Van Besien, W.L.: Dynamic, non-interactive key management for the bundle protocol. In: Proceedings of the 5th ACM Workshop on Challenged Networks. p. 75–78. CHANTS '10, Association for Computing Machinery (2010). https://doi.org/10.1145/1859934.1859951
14. Viswanathan, K., Templin, F.: Architecture for a Delay-and-Disruption Tolerant Public-Key Distribution Network (PKDN). Internet-draft, Internet Engineering Task Force (Mar 2016), https://datatracker.ietf.org/doc/draft-viswanathan-dtn-pkdn/00/