SDLP: A Lightweight Protocol for Authenticated Deep Links with Decentralized Identity

Prem Pillai
AI Engineer
Melbourne, Australia
prem.pillai@gmail.com

***Abstract*—Deep links enable seamless inter-application communication by encoding data directly within URLs, but they lack inherent security mechanisms to verify sender authenticity or payload integrity. This paper introduces the Secure Deep Link Protocol (SDLP), a lightweight protocol that provides cryptographic authentication and integrity verification for data transmitted via deep links. SDLP leverages JSON Web Signatures (JWS) with Ed25519 cryptography and Decentralized Identifiers (DIDs) to create verifiable links without requiring centralized certificate authorities. Our protocol achieves 72% payload efficiency within typical URL length constraints and supports flexible compression algorithms. Performance evaluation shows sub-millisecond link creation (0.09-0.11ms average) and verification times (0.32-0.38ms average) with throughput exceeding 9,000 operations per second. SDLP addresses critical security gaps in deep link communication while maintaining compatibility with existing URL schemes and mobile application ecosystems.**

***Index Terms*—deep links, authentication, decentralized identity, mobile security, cryptographic protocols, JSON Web Signature**

I. Introduction

Deep links serve as a fundamental mechanism for inter-application communication in modern computing environments, particularly in mobile ecosystems where they enable applications to invoke specific functionalities with contextual data [1], [2]. However, standard deep links present significant security vulnerabilities: they lack mechanisms to verify the sender's authenticity or ensure payload integrity, making them susceptible to spoofing, injection attacks, and data manipulation.

Consider these attack scenarios: A malicious actor could craft deep links containing harmful prompts for AI applications, manipulate configuration parameters in development tools, or inject unauthorized commands into automation systems. These vulnerabilities become particularly concerning as deep links increasingly carry sensitive data such as API tokens, user preferences, or executable configurations.

Existing solutions like JWT tokens embedded in URLs suffer from substantial overhead (approximately 78% size penalty due to double Base64URL encoding), while traditional PKI-based approaches require centralized certificate authorities that may not be available or appropriate for peer-to-peer scenarios.

This paper presents the Secure Deep Link Protocol (SDLP), which addresses these limitations through the following contributions:

1) A lightweight protocol design that separates cryptographic metadata from payload data to minimize URL overhead
2) Integration with Decentralized Identifiers (DIDs) for sender authentication without centralized authorities
3) Comprehensive performance evaluation demonstrating sub-millisecond operation latencies
4) Open-source reference implementation with production-ready benchmarking suite

II. Related Work

A. Deep Link Security

Traditional deep linking mechanisms in mobile platforms rely on custom URL schemes (iOS) or intent filters (Android) without built-in authentication [3], [4]. While Universal Links [2] and App Links [1] provide domain-based verification, they require web server control and don't address payload integrity.

Recent work has explored authenticated deep links for specific use cases. OAuth 2.0 for native applications [5] uses deep links for authorization flows but focuses on token exchange rather than general payload transmission. Mobile app security frameworks [6] identify deep link vulnerabilities but don't provide comprehensive solutions.

B. JSON Web Signatures and Tokens

JSON Web Signatures (JWS) [7] and JSON Web Tokens (JWT) [8] provide established patterns for cryptographic authentication of JSON payloads. However, their direct application to URL transmission suffers from encoding inefficiency: embedding a payload within JWT requires Base64URL encoding the payload, then encoding the entire JWT again for URL transmission, resulting in approximately 78% overhead.

C. Decentralized Identity

Decentralized Identifiers (DIDs) [9] enable cryptographic identity verification without centralized authorities. DID methods like `did:key` [10] and `did:web` [11] provide different approaches to key distribution and resolution. Our protocol leverages DIDs to enable sender identification in peer-to-peer scenarios where traditional PKI may not be feasible.

III. Protocol Design

A. Design Requirements

SDLP addresses the following requirements:

Authenticity: Recipients must be able to verify the sender's identity through cryptographic means.

Integrity: Any modification to the transmitted payload must be detectable.

Efficiency: The protocol must maximize useful payload capacity within URL length constraints.

Flexibility: Support for various payload types and compression algorithms.

Decentralization: No dependency on centralized certificate authorities.

Interoperability: Compatibility with existing URL schemes and mobile platforms.

B. Protocol Architecture

An SDLP link follows the structure:

```
<scheme>://<metadata>.<payload>
```

The protocol separates cryptographic metadata from the actual payload to avoid double encoding overhead. The metadata portion contains a Base64URL-encoded JWS object with sender information and payload verification data. The payload portion contains the Base64URL-encoded, optionally compressed actual data.

C. Metadata Structure

The metadata portion contains a JWS Flattened JSON Serialization with the following components:

Protected Header: Contains the cryptographic algorithm (`alg`) and key identifier (`kid`). The `kid` field specifies a complete DID URL pointing to a verification method in the sender's DID document.

Payload (Core Metadata): Contains protocol version (`v`), sender identifier (`sid`), payload MIME type (`type`), compression algorithm (`comp`), SHA-256 checksum (`chk`), and optional temporal bounds (`exp`, `nbf`).

Signature: Ed25519 signature over the JWS signing input, computed as ASCII(BASE64URL(protected) + '.' + BASE64URL(payload)).

D. Security Model

SDLP provides the following security guarantees:

Authentication: The DID resolution process and signature verification establish sender identity. Receivers can distinguish between known, trusted DIDs and unknown ones.

Integrity: The SHA-256 checksum in the core metadata, protected by the JWS signature, ensures payload integrity. Any modification to the payload will result in checksum mismatch.

Non-repudiation: Ed25519 signatures provide non-repudiation assuming proper private key protection.

The protocol does not provide:

Confidentiality: Payloads are transmitted in plaintext (though optionally compressed). Applications requiring confidentiality should implement encryption at the payload level.

TABLE I: Link Creation Performance

Payload Size	Average Time (ms)	Throughput (ops/sec)
32B (uncompressed)	0.09	11,600
256B (uncompressed)	0.11	9,200
1KB (uncompressed)	0.10	9,800
256B (Brotli)	0.11	9,000
1KB (Brotli)	0.11	9,100

Replay Protection: While optional temporal bounds (`exp`, `nbf`) provide basic replay protection, applications with stronger requirements should implement additional measures.

IV. Implementation

A. Reference Implementation

We developed a production-ready TypeScript SDK providing SDLP link creation and verification capabilities. The implementation includes:

- Ed25519 cryptographic operations using the `@noble/ed25519` library
- DID resolution supporting `did:key`, `did:web`, and extensible architecture for additional methods
- Compression support for Brotli, Gzip, and Zstandard algorithms
- Comprehensive error handling with standardized error codes
- CLI tools for testing and integration

The implementation prioritizes security through input validation, safe parsing, and protection against common vulnerabilities like JSON injection and malformed data attacks.

B. Payload Capacity Analysis

SDLP achieves significant efficiency improvements over naive JWT-in-URL approaches through architectural optimization:

Traditional JWT Approach: Payload $\rightarrow$ Base64URL $\rightarrow$ JWT $\rightarrow$ Base64URL results in $\approx 1.78\times$ size multiplication (78% overhead).

SDLP Approach: Payload $\rightarrow$ Compression $\rightarrow$ Base64URL results in $\approx 1.33\times$ size multiplication (33% overhead).

For a 32KB URL limit with estimated 1KB metadata overhead, SDLP provides approximately 23KB of compressed payload capacity, achieving 72% payload efficiency.

V. Performance Evaluation

A. Experimental Setup

We evaluated SDLP performance using a comprehensive benchmark suite on macOS ARM64 with Node.js v23.7.0. Tests measured link creation, verification, and capacity utilization across different payload sizes (32B, 256B, 1KB) with and without compression.

TABLE II: Link Verification Performance

Payload Size	Average Time (ms)	Throughput (ops/sec)
32B	0.32	3,100
256B	0.35	2,900
1KB	0.38	2,600

TABLE III: Compression Analysis

Payload	Uncompressed	Compressed	Reduction
32B	789B total	N/A	0%
256B	1,117B total	1,009B total	9.7%
1KB	2,233B total	1,433B total	35.9%

B. Link Creation Performance

Link creation performance remains consistent across payload sizes:

The consistent sub-millisecond creation times demonstrate that cryptographic operations dominate performance rather than payload processing, making SDLP suitable for real-time applications.

C. Verification Performance

Link verification shows slightly higher latency due to DID resolution and signature validation:

These results include DID document parsing and Ed25519 signature verification. In production scenarios, DID document caching could further improve performance.

D. Compression Effectiveness

Brotli compression provides significant space savings for larger payloads:

Compression effectiveness increases with payload size, with 1KB payloads achieving 35.9

VI. Security Analysis

A. Threat Model

SDLP addresses threats from attackers who can:

- Intercept and modify deep links in transmission
- Generate malicious deep links impersonating legitimate senders
- Attempt replay attacks using previously valid links

SDLP does not address threats from:

- Compromised sender private keys
- Malicious payload content (applications must implement content validation)
- Side-channel attacks on cryptographic implementations

B. Cryptographic Security

SDLP leverages established cryptographic primitives:

Ed25519: Provides 128-bit security level with resistance to timing attacks and deterministic signatures [12].

SHA-256: Ensures collision resistance for payload integrity verification [13].

JWS: Follows RFC 7515 for signature format and verification procedures.

C. DID Security Considerations

Different DID methods provide varying security properties:

did:key: Self-contained keys provide immediate verification but lack revocation mechanisms.

did:web: Web-based keys enable updates and revocation but depend on DNS and HTTPS security.

Applications should choose DID methods appropriate to their security requirements and threat models.

VII. Applications and Use Cases

SDLP addresses security requirements across various application domains:

AI Prompt Sharing: Authenticated prompts prevent injection attacks and ensure prompt provenance in AI applications.

Configuration Distribution: Development tools can securely distribute configuration data with verified authenticity.

Cross-Application Authentication: Applications can exchange authentication tokens through verified deep links.

IoT Device Control: Authenticated commands prevent unauthorized device manipulation.

Data Sharing: Peer-to-peer applications can exchange small data objects with integrity guarantees.

VIII. Limitations and Future Work

Current limitations include:

Key Management: Applications must implement secure private key storage and protection mechanisms.

DID Resolution Dependencies: Network-based DID methods require connectivity and may introduce latency.

URL Length Constraints: While optimized, SDLP still faces fundamental URL length limitations across different platforms.

Future work directions include:

Standardization: Pursuing formal standardization through appropriate standards bodies.

Additional DID Methods: SDK support for emerging DID methods like `did:peer` for peer-to-peer scenarios (protocol already supports any W3C DID method).

Advanced Features: Investigating selective disclosure, zero-knowledge proofs, and enhanced privacy features.

Performance Optimization: Exploring cryptographic algorithm alternatives and implementation optimizations.

IX. Conclusion

The Secure Deep Link Protocol addresses critical security gaps in deep link communication through a lightweight, efficient design that provides authentication and integrity verification without centralized dependencies. Our performance evaluation demonstrates that SDLP achieves sub-millisecond operation latencies while maintaining 72% payload efficiency within URL constraints.

SDLP's integration with Decentralized Identifiers enables trustless authentication scenarios while maintaining compatibility with existing mobile platforms and URL schemes. The protocol's flexibility in supporting various payload types

and compression algorithms makes it suitable for diverse applications from AI prompt sharing to IoT device control.

The open-source reference implementation and comprehensive benchmarking suite facilitate adoption and further research. As deep links become increasingly prevalent for inter-application communication, SDLP provides a foundation for secure, authenticated data transmission that scales from simple configuration sharing to complex multi-party scenarios.

ACKNOWLEDGMENTS

The author thanks the W3C DID Working Group for their work on Decentralized Identifier specifications, and the broader cryptographic community for developing the foundational primitives that enable secure protocols like SDLP.

REFERENCES

[1] Google, "Android app links," 2023, android Developer Documentation. [Online]. Available: https://developer.android.com/training/app-links

[2] Apple, "Universal links," 2023, iOS Developer Documentation. [Online]. Available: https://developer.apple.com/ios/universal-links/

[3] Google, "Intent and intent filters," 2023, android Developer Documentation. [Online]. Available: https://developer.android.com/guide/components/intents-filters

[4] Apple, "Custom url schemes," 2023, iOS Developer Documentation. [Online]. Available: https://developer.apple.com/documentation/xcode/defining-a-custom-url-scheme-for-your-app

[5] W. Denniss and J. Bradley, "Oauth 2.0 for native apps," October 2017, rFC 8252. [Online]. Available: https://tools.ietf.org/rfc/rfc8252.txt

[6] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *ACM Conference on Computer and Communications Security*. ACM, 2013, pp. 73–84.

[7] M. Jones, J. Bradley, and N. Sakimura, "Json web signature (jws)," May 2015, rFC 7515. [Online]. Available: https://tools.ietf.org/rfc/rfc7515.txt

[8] ——, "Json web token (jwt)," May 2015, rFC 7519. [Online]. Available: https://tools.ietf.org/rfc/rfc7519.txt

[9] W3C, "Decentralized identifiers (dids) v1.0," July 2022, w3C Recommendation. [Online]. Available: https://www.w3.org/TR/did-core/

[10] D. Longley, D. Zagidulin, and M. Sporny, "The did:key method v0.7," 2021, w3C Community Group Report. [Online]. Available: https://w3c-ccg.github.io/did-method-key/

[11] O. Steele, M. Sporny, and D. Longley, "The did:web method v0.1," 2021, w3C Community Group Report. [Online]. Available: https://w3c-ccg.github.io/did-method-web/

[12] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012.

[13] NIST, "Secure hash standard (shs)," August 2015, fIPS PUB 180-4.