岐阜大学工学部

電気電子・情報工学科

令和 6 年度卒業論文

セキュアな V2V アドホックネットワーク ルーティングプロトコルのための EdDSA 署名方式の評価

三嶋研究室

学籍番号：1213033107

永野 正剛

指導教員

三嶋 美和子 教授

角田 有 助教

目次

はじめに

はじめにを書くよ

第 1 章　準備

この章では, 本研究で使用する通信技術について説明する. 1.1 節では, VANET について, 1.2 節では VANET で用いられるルーティングプロトコルである GPSR について述べる. 1.3 節では, VANET のセキュリティを担保するために使用するデジタル署名について解説する.

1.1　Vehicle Ad Hoc Network

VANET (Vehicle Ad Hoc Network) とは, モバイルアドホックネットワーク技術を車両間通信に応用したネットワークである [**?**, **?**]. VANET は, 車両間の通信 (Vehicle-to-Vehicle, **V2V**), および車両インフラ (路側機やドローン) 間通信 (Vehicle-to-Infrastructure, **V2I**)[**?**] で構成され, 固定されたインフラに依存せず, ノード同士が自律的に通信ネットワークを形成する. VANET では, 車両間の距離が無線通信の範囲を超えることが一般的であるため, 図 1.1 のように, データを送信元から宛先まで直接通信できない場合に, 中継ノードを経由してデータを転送する. このような通信を**マルチホップ通信**という.

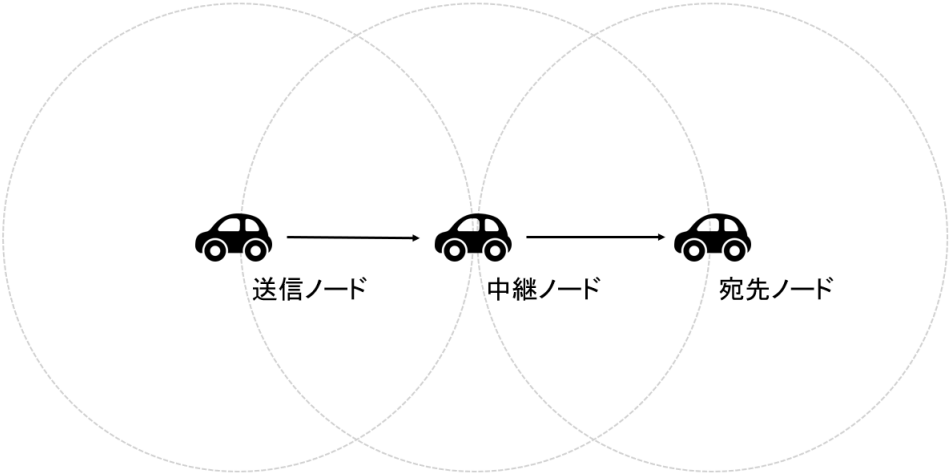

図 1.1　マルチホップ通信 [?]

VANET には次の 5 つの特徴がある.

(1) 十分な電力供給

VANET を利用する際, 車両は走行していることを前提とするため, スマートフォンのような電池駆動のモバイルデバイスとは異なり, 電力の制約は無視できる. したがって, 長時間の

稼働や高い通信レートの実現が可能であると仮定する．とはいえ, 高性能な通信モジュールやセンサーを多数搭載する場合, 車両のエネルギー効率に影響を与える可能性が考えられるため, 依然として効率的なデバイス設計は求められる．

(2) 位置情報の取得

車両は GPS を搭載しているため, 自身の位置情報を取得できる．

(3) ネットワークトポロジーの急速な変化

無線通信では，ノード同士が直接的に通信可能であることを**接続している**といい，この接続状態をもとにネットワーク全体の構造が形成される． このネットワーク構造を**ネットワークトポロジー**という． VANET では通信するノードを車両と想定しているため, 移動速度の速いノードが動的にネットワークを形成する． そのため, 接続の頻繁な確立と切断が発生し, ネットワークトポロジーは急速に変化する．

(4) 移動の制約

車両の動きは道路や建造物などの物理的構造に従う．

(5) 安全に関する情報のリアルタイム性

交通事故や道路状況に関する情報を即座に共有するためには, 低遅延かつ信頼性の高い通信が求められる．

VANET は, 車両間通信や車両インフラ間通信を実現するための優れた技術としてのポテンシャルを有する一方で, いくつかの課題も抱えている．その中でも, セキュリティに関する課題は, VANET を安全かつ信頼性の高いシステムとして運用する上で最も重要な課題の 1 つとなっている [?, ?].

1.2 Greedy Perimeter Stateless Routing

GPSR (Greedy Perimeter Stateless Routing)[?] は, 位置情報を利用してパケットを転送する位置ベースのルーティングプロトコルであり, VANET のような動的かつ高速に変化するネットワーク環境に適している． GPSR では, 自身の位置や IP アドレスなどの情報をのせた Hello パケットを一定間隔で隣接ノードに送信する．図 1.2 に示すように, それぞれのノードは IP アドレスや隣接ノードの位置などの情報が記載された隣接ノードテーブルをもち, 受信した Hello パケットの情報を用いて隣接ノードテーブルを更新することにより, 周辺ノードの情報を把握する．この隣接ノードテーブルの情報をもとに, Greedy Forwarding と Perimeter

Forwarding を組み合わせたルーティングプロトコルに従う.

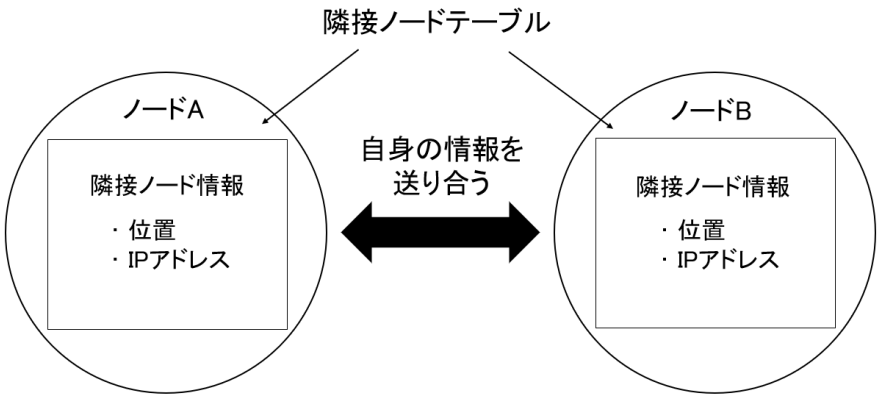

図 1.2　GPSR の隣接ノードテーブル [?]

Greedy Forwarding

Greedy Forwarding は GPSR の基本的なルーティングプロトコルである. 図 1.3 に示すように, 送信ノード S は, 宛先ノード D の位置情報をもとに自身の電波伝搬範囲内のノードから, D に最も近いノードをネクストホップとして選択する. 前提として, 送信ノード S は宛先ノード D の位置情報を事前に把握しているものとする. 点線で書かれた円は全ノードの受信感度が等しい場合の送信ノード S の電波伝搬範囲を, 破線は宛先ノードとの距離を表している.

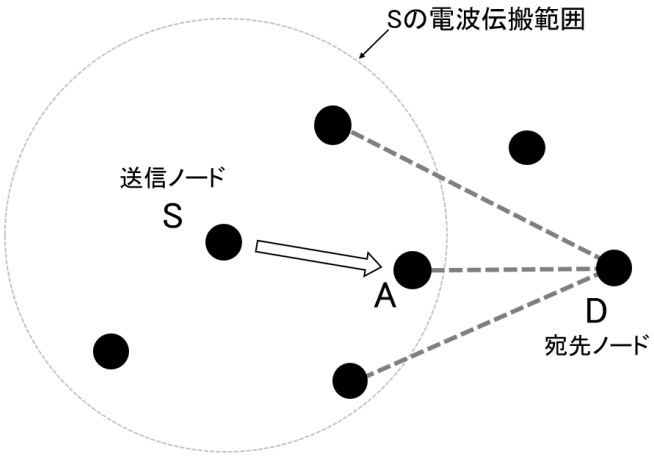

図 1.3　Greedy Forwarding[?]

Greedy Forwarding では局所最大問題と呼ばれる問題が生じてしまうことがある. **局所最大問題**とは, 図 1.4 に示すように, 送信ノード S の電波伝搬範囲内に宛先ノード D が存在しない,

かつ, 送信ノード S が自身の電波伝搬範囲内で宛先ノード D に最も近い場合, 選択できるネクストホップが存在しなくなるという問題である.

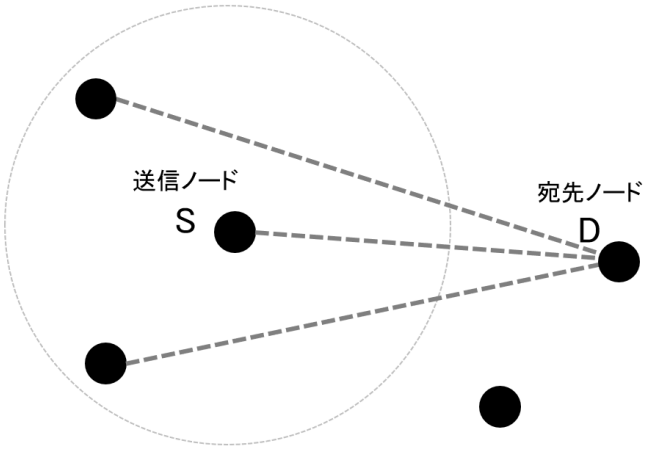

図 1.4　局所最大問題

Perimeter Forwarding

Greedy Forwarding で局所最大問題が発生した場合に, Perimeter Forwarding が使用される. **Perimeter Forwarding** とは, 図 1.5 に示すように, 送信ノード S を中心にして宛先ノードの方向から, 反時計回りにノードを探索し, 最初に発見したノードをネクストホップとして選択する方式である.

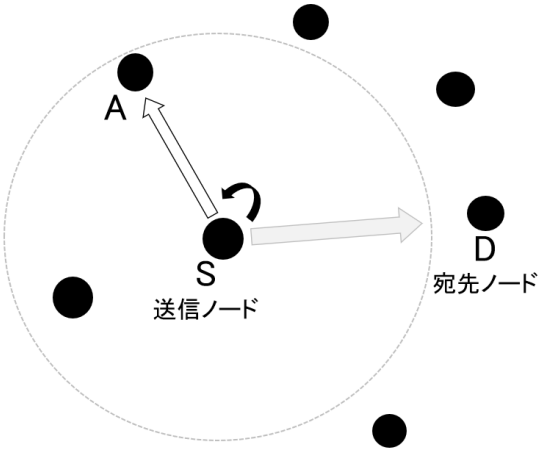

図 1.5　Perimeter Forwarding[?]

1.3 デジタル署名

デジタル署名とは, 公開鍵暗号の仕組みを利用して作られた暗号技術であり, 電子文書やメッセージの真正性と完全性を検証するために使用される. この技術は, データの信頼性を確保し, かつ, 送信者がデータの送信を否定できない非否認性を提供する.

一般に, デジタル署名は以下の 3 つのフェーズから構成される.

1. **鍵生成**
   署名者（送信者）が署名に使用する鍵のペア, すなわち検証鍵 (公開鍵) と署名鍵 (秘密鍵) を生成するフェーズである. 署名鍵は署名者が厳重に管理し, 秘密に保管する. 一方, 検証鍵は受信者や第三者に公開され, 署名の検証に用いられる.
2. **署名生成**
   メッセージ作成者 (送信者) は, 署名対象のメッセージからハッシュ関数を用いてハッシュ値を計算する. このハッシュ値は, メッセージがわずかでも変更されると全く異なる値となるため, メッセージが改ざんされていないこと (完全性) を検証するための重要な要素である. 続いて, メッセージ作成者は自身の署名鍵 (秘密鍵) を使ってハッシュ値に署名を行う. 署名鍵がメッセージ作成者しか保持していないことと, メッセージ作成者以外が署名鍵を求めることは計算量的に困難なことから, 署名はメッセージの作成者が正当であること (真正性) を証明する役割を果たす.
3. **署名検証**
   検証者（受信者）は, 署名者が公開した検証鍵 (公開鍵) を用いて署名を検証する. この検証プロセスにより, 署名が署名鍵と対になる検証鍵によって生成されたことを確認できるため, メッセージが正規の作成者によって署名され, 送信後に改ざんされていないことが保証される.

デジタル署名にはさまざまなアルゴリズムが存在する. その中でも, **国立標準技術研究所 (National Institute of Standards and Technology, NIST)** による情報処理標準規格 FIPS に基づいて設計された **DSA(Digital Signature Algorithm)** およびその改良版である **ECDSA(Elliptic Curve Digital Signature Algorithm)** は, V2V 通信のようなリアルタイム性やリソース制約が求められる環境に適している. したがって, 本研究でも先行研究 [?] と同様に, 2 つのデジタル署名方式を実装する.
以下に, DSA と ECDSA の概要を示す.

DSA

DSA (Digital Signature Algorithm) は, 1993 年に FIPS 186-4 として標準化された, DSS (Digital Signature Standard) の主要なアルゴリズムの 1 つであった. なお, 2023 年の FIPS 186-5[?] では, DSA は新たにデジタル署名を行うことには推奨されないが, 標準策定以前に行われた署名の検証には引き続き利用可能とされている.

DSA の 3 つのフェーズにおける処理は以下の通りである.

鍵生成

1. セキュリティパラメータとして整数 $L, N(L > N)$ を定める. FIPS 186-4 では以下の 4 つの値の組が規定されている.

$$(L, N) = (1024, 160), (2048, 224), (2048, 256), (3072, 256)$$

2. L ビットのランダムな素数 $p(2^{L-1} < q < 2^L)$, N ビットのランダムな素数 $q(2^{N-1} < q < 2^N)$ を選ぶ. ただし, q は $p-1$ を割り切る素数とする.

3. ランダムな整数 $a < p-1$ に対し,

$$g \equiv a^{\frac{p-1}{q}} \pmod{p}$$

を計算する. ただし, $g = 1$ である場合は再度 a を選び直す.

4. ランダムな整数 $x(1 < x < q)$ に対し,

$$y \equiv g^x \pmod{p}$$

を計算する.

5. p, q, g は専用のパラメータとして, y は検証鍵 (公開鍵) として公開する. x は署名鍵 (秘密鍵) として安全に管理する.

任意長のメッセージ M に対して, パラメータ p, q, g, 秘密鍵 x, ハッシュ関数 H を用いて, 次のように署名を生成する.

署名生成

1. ランダムな整数 $k(1 < k < q)$ を選ぶ.

2. $r \equiv (g^k \pmod{p}) \pmod{q}$ を計算する. ただし, $r = 0$ の場合は再度 k を選び直す.

3. $s \equiv k^{-1}(H(M+xr)) \pmod{q}$ を計算する. ただし, $s=0$ の場合は再度 k を選び直す.
4. (r,s) をメッセージ M に対する署名とし, M とともに受信者に送信する.

メッセージ M に対する署名 (r,s) を検証するには, パラメータ (p,q,g), 公開鍵 y, ハッシュ関数 H を用いて, 以下のアルゴリズムを実行する.

署名検証

1. 受け取った (r,s) から, $0<r<q$ かつ $0<s<q$ であることを確認する. これを満たさない場合は署名を棄却する.
2. $w \equiv s^{-1} \pmod{q}$ を計算する.
3. $u_1 \equiv wH(M) \pmod{q}$, $u_2 \equiv rw \pmod{q}$ を計算する.
4. $v \equiv (g^{u_1}y^{u_2} \pmod{p}) \pmod{q}$ を計算する.
5. $v \equiv r \pmod{q}$ であれば, 署名を受理する. そうでなければ不正な署名とみなし, 棄却する.

正当なメッセージと署名の組 $(M,(r,s))$ に対し,

$$g^{u_1}y^{u_2} \equiv g^{u_1+xu_2} \equiv g^{H(M)+xr}w \equiv g^k \pmod{p}$$

が成り立つ. したがって, 正当なメッセージと署名に組には

$$\begin{aligned} v &\equiv (g^{u_1}y^{u_2} \mod p) \mod q \\ &\equiv (g^k \mod p) \mod q \\ &\equiv r \mod q \end{aligned}$$

が成り立つため, $v \equiv r$ を満たすかどうかを確認することで署名検証が可能である.

ここで, メッセージや署名に対し改ざんが行われたとしよう. 攻撃者が, 署名検証条件 $v \equiv r \pmod{q}$ を満たすように操作できた場合, その攻撃は成功したとみなされる. ただし, 署名生成に使用される値には, 署名者しか知らない秘密の乱数 k が含まれており, この k は離散対数問題に基づいて計算されるため, 十分なビット長を持つ p および q の下では, 改ざんを成功させることは計算量的に極めて困難である. さらに, 署名生成プロセスではメッセージのハッシュ値 $H(M)$ が使用されており, このハッシュ値の安全性はハッシュ関数の衝突困難性に依存している. したがって, 攻撃者が異なるメッセージ M' を生成し, そのハッシュ値が元

のメッセージ M と同じ $H(M') = H(M)$ となるようにすることも困難である．この性質により, 改ざんされたメッセージ M' のハッシュ値は $H(M') \neq H(M)$ となり, 署名検証条件 $v \equiv r \pmod{q}$ が成立しなくなる．また, 攻撃者がメッセージ M をそのままにして署名 (r, s) を改ざんした場合でも, 署名にはハッシュ値 $H(M)$ が埋め込まれているため, 署名検証条件は満たされない． この結果, 署名検証アルゴリズムの最終段階で署名が有効と判定される場合, 署名 (r, s) もメッセージ M も改ざんされていないことが保証される．

ECDSA

ECDSA (Elliptic Digital Signature Algorithm) は, DSA を楕円曲線暗号（Elliptic Curve Cryptography, ECC）を基盤として改良したデジタル署名アルゴリズムである． ECDSA は, 2000 年に FIPS 186-2 として標準化され, 最新版である FIPS 186-5[?] でも採用されている．ECDSA は, DSA よりも短い鍵長で同等の安全性を確保できるため, 計算量が少なく, 低性能なデバイスでも高速な処理が可能であり, かつ演算規則の複雑さから攻撃者が鍵を推測することも困難である．よって, 処理速度と安全性において有限体上の DSA よりも優れている．

この項では, ECDSA の理解に必要となる楕円曲線とその上での演算について説明し, その後, ECDSA のアルゴリズムについて述べる．

楕円曲線

体 $\mathbb{F}_p$(p は素数) 上で定義された楕円曲線とは以下の式で与えられる代数曲線である．

$$y^2 + a_1 y + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \qquad (a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_p)$$

この式を Weierstrass 方程式という [?]．特に, $p \neq 2, 3$ の場合, 以下の標準形に簡略化できることが知られている．

$$y^2 = x^3 + ax + b \qquad (a, b \in \mathbb{F}_p) \tag{1.1}$$

楕円曲線 (式 (1.1)) の判別式 Δ が

$$\Delta = -16(4a^3 + 27b^2) \neq 0$$

を満たすとき, 楕円曲線 (式 1.1) は**非特異**であるという．非特異である楕円曲線は, 尖点や自己交差点, 孤立点を持たないため, 楕円曲線が非特異であることは, 後述する楕円曲線上の点に対する演算 (加算やスカラー倍) が矛盾なく定義されるために必要不可欠である．

楕円曲線上の点 $P = (x, y)$ のうち x, y がともに有限体 $\mathbb{F}_p$ の元であるものを**有理点**という．また, 楕円曲線上には**無限遠点**と呼ばれる特殊な点 $\mathcal{O}$ の存在を仮定し, これも楕円曲線上の有理点の集合に含める．有理点の集合は, 以下で述べる演算に関して群の構造をもつ．楕円曲

線暗号方式では，この性質が利用される.

楕円曲線上の点の加算

楕円曲線上の点 $P=(x_1,y_1)$ と $Q=(x_2,y_2)$ の加算 $P+Q$ を以下のように定義する.

(i) 2 点 P,Q を通る直線が y 軸と平行でない場合 (図 1.6)：

1. 点 P,Q を通る直線 L を引く.

$$L: y-y_1=\frac{y_2-y_1}{x_2-x_1}(x-x_1)$$

2. L と楕円曲線 E の交点を $R(x_3,y_3)$ とする.

$$x_3=\left(\frac{y_2-y_1}{x_2-x_1}\right)^2-x_1-x_2$$
$$y_3=\frac{y_2-y_1}{x_2-x_1}(x_1-x_3)-y_1$$

3. R の x 軸対称な点 $R'(x_3,-y_3)$ が $P+Q$ となる.

$$P+Q=R'$$

図 1.6 を挿入

(ii) 2 点 P,Q を通る直線 L が y 軸と平行である場合：

この場合, 直線 L は 2 点 P,Q 以外で楕円曲線 E と交わらない.

無限遠点 $\mathcal{O}$ が $P+Q$ となる.

$$P+Q=\mathcal{O}$$

(iii) 2 点 P,Q が同一の点である場合 (図 1.7)：

1. 点 P における接線 L' を引く.

$$L': y-y_1=\frac{3x_1^2+a}{2y_1}(x-x_1)$$

2. L' と楕円曲線 E の交点を $R(x_3,y_3)$ とする.

$$x_3=\left(\frac{3x_1^2+a}{2y_1}\right)^2-2x_1$$
$$y_3=\frac{3x_1^2+a}{2y_1}(x_1-x_3)-y_1$$

3. R の x 軸対称な点 $R'(x_3,-y_3)$ が $P+Q$ となる.

$$P+Q=2P=R'$$

図1.7を挿入

(iv) $P = \mathcal{O}$ または $Q = \mathcal{O}$ である場合：

無限遠点は加算において加法単位元の役割を果たす.

$$P + \mathcal{O} = P$$

$$\mathcal{O} + Q = Q$$

スカラー倍算

正整数 k による点 G のスカラー倍 $P = kG$ は, 点 G を k 回加算した結果を表す.

$$P = kG = \underbrace{G + G + \cdots + G}_{k\text{回}}$$

ある基準点 G に対し, $P = kG$ となる楕円曲線上の点 P が与えられたとき, k と G から P を求めるのは容易である. しかし, 逆に G と P から k を求めるのは計算量的に困難であることが知られている. これを**楕円曲線上の離散対数問題 (Elliptic Curve Discrete Logarithm Problem, ECDLP)** という.

ECDSA のアルゴリズム

ECDSA の 3 つのフェーズにおける処理は以下の通りである.

鍵生成

1. 法となる素数 p と, 楕円曲線 E を選び, E 上の基準点 G を選ぶ.
2. d を $2 \leq d \leq n-1$ の範囲からランダムに選び, 署名鍵 (秘密鍵) として安全に管理する. ただし, n は G の位数である.
3. $Q = dG$ を計算し, Q を検証鍵 (公開鍵) とする.

署名生成

1. k を $2 \leq d \leq n-1$ の範囲からランダムに選び, kG の x 座標を r とする.
2. メッセージ M に対し, ハッシュ関数 H を用いて, $h = H(M)$ を計算する.
3. $s \equiv k^{-1}(h + dr) \pmod{n}$ を計算する.
4. (r, s) をメッセージ M に対する署名とし, M とともに受信者に送信する.

署名検証

1. $(M, (r, s))$ を受け取り, $h = H(M)$ を計算する.
2. $u \equiv s^{-1}h \pmod{n}$, $v \equiv s^{-1}r \pmod{n}$ を計算する.
3. 楕円曲線上の点として,
$$Q' = (x', y') = uG + vQ$$
を計算する.
4. Q' の x 座標 x' が r と一致すれば, 署名を受理する. そうでなければ不正な署名とみなし, 棄却する.

正当なメッセージと署名の組 $(M, (r, s))$ に対し,

$$\begin{aligned} Q' &= uG + vQ \\ &= s^{-1}hG + s^{-1}rQ \\ &= s^{-1}hG + s^{-1}rdG \\ &= s^{-1}hG + s^{-1}(sk - h)G \\ &= s^{-1}(hG + skG - hG) \\ &= s^{-1}skG \\ &= kG \end{aligned}$$

が成り立つため, Q' の x 座標が r と一致するかどうかを確認することで署名検証が可能である.

第 2 章　関連研究

階戸 [?] は, 既存のルーティングプロトコル GPSR を改良し, 車両の位置情報と IP アドレスの詐称による不正を防ぐため, デジタル署名による認証を導入した. 以下, この改良版を「階戸のプロトコル」と呼ぶ. この章では, 階戸のプロトコルについて解説する.

階戸のプロトコル

位置情報を活用したルーティングプロトコルである GPSR が正しく機能するためには, 同一アドホックネットワーク内の参加者が相互に正しい位置情報を送信する必要がある. しかし, ネットワーク内に不正なノードが存在する場合, ルーティングが妨害されかねない. この章では, 想定される不正ノードを示した後, その対策について述べる.

不正ノード

1.　位置情報詐称ノード

位置情報詐称ノードとは, 通信データの窃取を目的に, 自身の位置情報を偽装してルーティングを妨害する内部不正ノードである. **内部不正ノード**とは, ネットワーク内において意図的に不正行為を行うノードのことで, 外部から侵入した攻撃者や, 内部の信頼されたノードが乗っ取られることで発生することが一般的である. 図 2.1 に示すように, 送信ノード S が宛先ノード D に送信するとする. このとき, 本来 S は電波伝搬範囲内で D に最も近い A を中継ノードとして選択するべきであるが, ノード B が位置情報を詐称して B' の位置にいると偽装することで, B を選択し, 誤ったルーティングをしてしまう. このようにして, B は本来 D が受けるべきデータを窃取し, 情報伝達を妨害できる.

2.　IP アドレス詐称ノード

IP アドレス詐称ノードとは, ネットワークへの不正アクセスを目的とした外部不正ノードである. **外部不正ノード**とは, ネットワークの外部から侵入し, 不正な目的でネットワーク内のデータやリソースを攻撃または利用しようとするノードのことである. 図 2.2 は, 当該ネットワークの構成ノードを黒丸, 非構成ノードを赤丸で示している. 赤丸で示した外部ノードがネットワーク内の IP アドレスを詐称してネットワーク参加者に送信した場合, その情報を受け取ったノードは外部ノードを信頼し, 通信を行ってしまう. このように, IP アドレスを詐称することで外部ノードがネットワークに不正に参加できてしまう.

このような 2 種類の不正ノードが存在する場合, 正しいルーティングが行われない. そこ

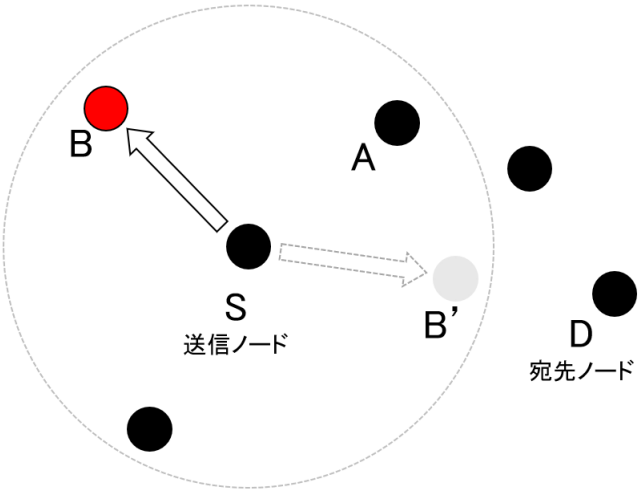

図 2.1　位置情報詐称ノード [?]

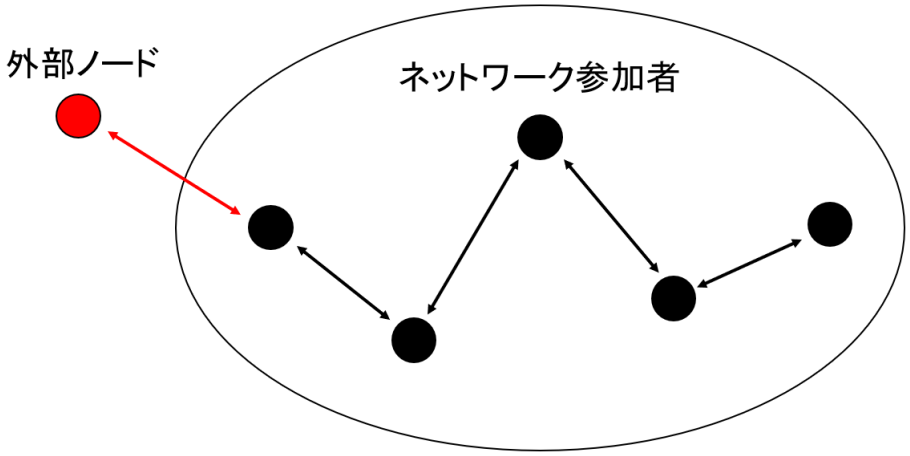

図 2.2　IP アドレス詐称ノード [?]

で，本研究では，デジタル署名を用いてノードの情報が正しいものであるかを検証し，なりすましや改ざんを排除するようにした．なお，一般にはメッセージ作成者 (送信者) が署名者であるのに対し，本研究では署名の送信者と署名者が異なる．そのため，送信者 (ノード) は署名の作成に関して携わらない．

では，その仕組みを説明する．図 2.3 に示すように，ノード A からノード B にデータを送信する場合，A は認証機構による署名をデータに付与して B に送信する．B は受信したデータに対して，認証機構による公開鍵を用いて署名の検証を行い，ネットワーク内の正当なノードから正しいデータが送信されたかを確認する．

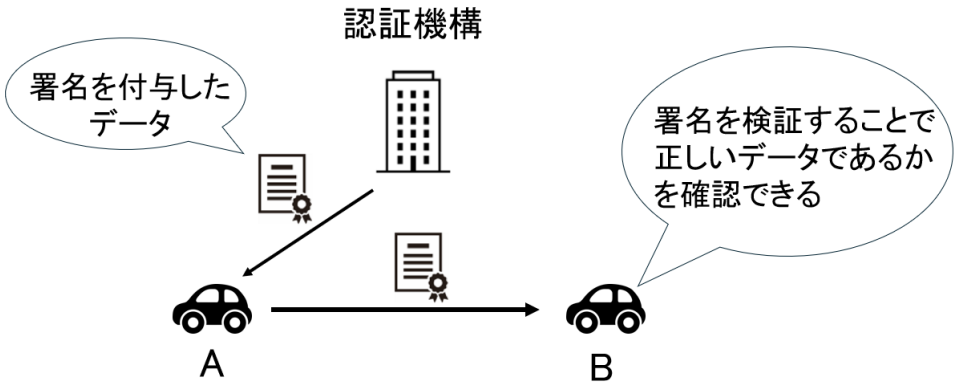

図 2.3　本研究におけるデジタル署名の使用方法 [?]

本研究で導入される認証機構は以下の通りである.

表 2.1: 認証機構 (署名者) と被署名データの対応

認証機構 (署名者)	被署名データ
DHCP サーバ	IP アドレス
GPS または位置情報が正しいと証明できる機関	位置情報

これらの認証機構による認証方法を次の手順で導入する (図 2.4).

1. 各ノードが DHCP サーバから IP アドレスを取得する際, DHCP サーバから署名をもらう.
2. 各ノードが GPS から位置情報を取得する際, GPS または位置情報が正しいと証明できる機関から署名をもらう.
3. 各ノードが隣接ノードと Hello パケットを交換する際, IP アドレスと位置情報の署名を一緒に送信する.
4. Hello パケットを受信したノードは 2 つの署名を検証し, どちらの署名も検証が成功した場合のみ, 隣接ノードテーブルを更新する. どちらか一方でも署名検証に失敗した場合, その Hello パケットを破棄し, 隣接ノードテーブルの更新を行わない.

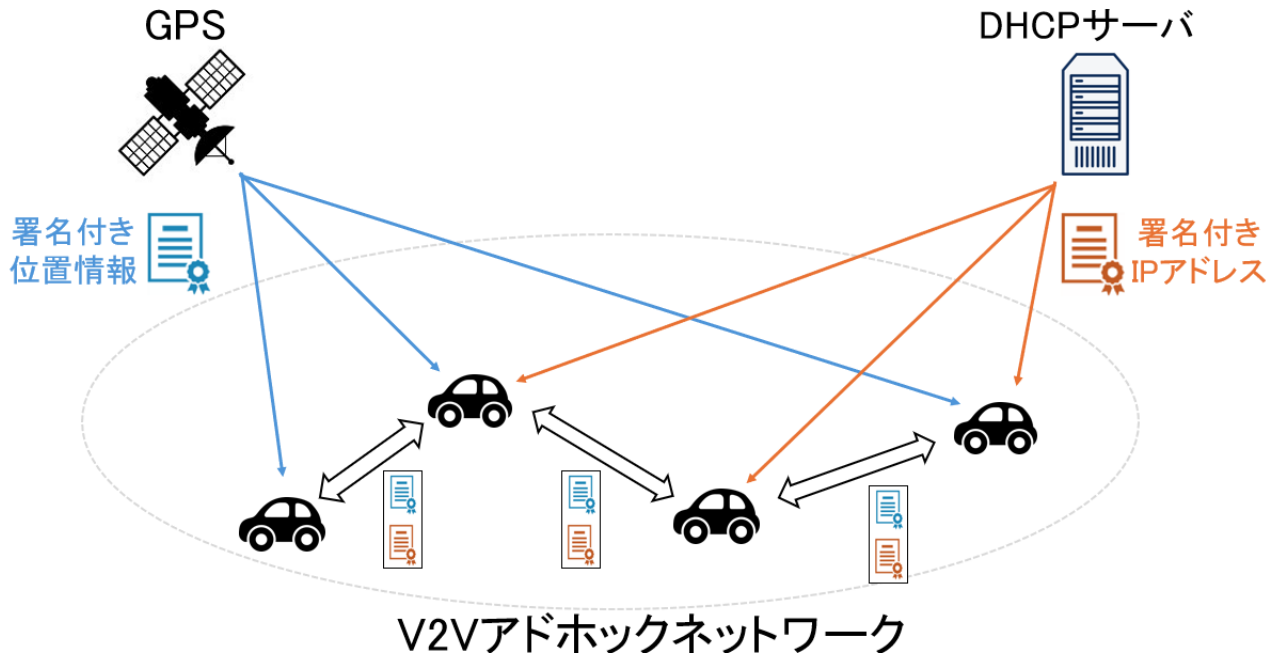

図 2.4　デジタル署名を用いた認証機構 [?]

第 3 章　EdDSA

EdDSA(**Edwards-curve Digital Signature Algorithm**) とは, 特殊な楕円曲線であるエドワーズ曲線, またはツイストエドワーズ曲線 [?] を利用したデジタル署名のことであり, 2011 年に Daniel J. Bernstein らによって提案された [?]. EdDSA は, ECDSA に比べて鍵長や署名長にはほとんど差がないが, 使用される曲線によって高速な演算を可能にし, 処理速度を向上させている. さらに, DSA をもとに設計された ECDSA と異なり, EdDSA はシュノア署名 [?] に基づいて設計されているため, シンプルかつ堅牢な構造を踏襲している (その詳細については 2.6 節で解説する). つまり, EdDSA は ECDSA の脆弱性を克服するような設計となっており, 高速な処理速度と高い安全性を兼ね備えている. この特徴が評価され, インターネットの技術的な標準を策定する国際的な組織 IETF によって EdDSA に関する公式な文書 RFC 8032 [?] が, 2017 年に公開された. また, 2019 年には NIST の FIPS 186-5 で標準化された. 加えて, EdDSA はインターネット上での安全な通信を確立するための TLS(Transport Layer Security)1.3 [?] やネットワーク経由で他のコンピュータと安全に通信するためのプロトコルである SSH, さらには, ブロックチェーン技術 [?] など, さまざまなプロトコルやアプリケーションで採用されている. このように, EdDSA は比較的新しいデジタル署名アルゴリズムでありながら, その安全性と効率性から, 広く利用されている. したがって, 本研究に EdDSA を採用することにより, 通信性能を向上させつつ, 高いセキュリティ基準を満たす署名方式を実現することが期待される.

RFC8032 によると, EdDSA では暗号攻撃に対して約 128 ビットのセキュリティレベルの **Ed25519** と, 約 224 ビットのセキュリティレベルの **Ed448** の 2 種類の実装のどちらかを安全性要件に応じて利用するよう推奨されている. Ed25519 は 2005 年に Bernstein らによって提案された Curve25519[?] を基盤としており, 特に「ツイストエドワーズ曲線」という形式を採用することで, 計算効率とセキュリティを向上させている. 現在, Ed25519 は EdDSA の最も一般的なインスタンスであり, 必要なリソースが少なく, 十分な安全性を提供している. Ed448 は Ed25519 よりもセキュリティレベルが高いが, オーバースペックとなり, 一般的な利用には向いていない. よって, 本研究では Ed25519 を用いてデジタル署名を実装する.

この章では Ed25519 について概説する. 2.1 節では Ed25519 の理解に必要となるツイストエドワーズ曲線とその上での演算について述べる. 2.2 節では Ed25519 で使用されるデータの変換について, 2.3 節では Ed25519 のパラメータについて述べる. 2.4 節では Ed25519 のデジタル署名アルゴリズムについて述べ, 最後に, 2.5 節では ECDSA と Ed25519 のセキュリティ

と計算効率の比較を行い, Ed25519 を用いる利点を説明する.

3.1 ツイストエドワーズ曲線

この節では EdDSA で使用される**ツイストエドワーズ曲線 (Twisted Edwards Curve)** について説明する.

エドワーズ曲線は, 体 $\mathbb{F}_p$ (p は素数かつ $p \neq 2$) 上で定義され, 元 $c, d \in \mathbb{F}_p, cd(cd^4 - 1) \neq 0$ に対し, 以下の式で与えられる.

$$x^2 + y^2 = c^2(1 + dx^2y^2) \tag{3.1}$$

この曲線は, 変数変換することで Weierstrass 方程式 (式 (1.1)) で定まる楕円曲線と同型であることが知られている [**?**].

ツイストエドワーズ曲線は体 $\mathbb{F}_p$ (p は素数かつ $p \neq 2$) 上で定義され, 異なる非零元 $a, d \in \mathbb{F}_p$ に対し, 以下の式で与えられる.

$$ax^2 + y^2 = 1 + dx^2y^2 \tag{3.2}$$

ツイストエドワーズ曲線は, 通常のエドワーズ曲線 (式 (3.1)) と拡大体上で同型である. この同型により, ツイストエドワーズ曲線上の点集合は群をなす [**?**].

ツイストエドワーズ曲線上の楕円加算

ツイストエドワーズ曲線上の点 (x_1, y_1) と (x_2, x_2) の加算を以下のように定義する.

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 + ax_1x_2}{1 - dx_1x_2y_1y_2} \right)$$

無限遠点

ツイストエドワーズ曲線で無限遠点 $\mathcal{O}$ は以下の性質をもつ.

(1) 加法における単位元

無限遠点は楕円曲線 E 上の特別な点で任意の点 P に対して, 次の式が成り立つ.

$$P + \mathcal{O} = \mathcal{O} + P = P$$

ツイストエドワーズ曲線を使用した Ed25519 において, 無限遠点 $\mathcal{O}$ は次の座標をもつと定義される.

$$\mathcal{O} = (0, 1)$$

曲線の方程式 (式 (3.2)) に $(x, y) = (0, 1)$ を代入すると,

$$a \cdot 0^2 + 1^2 = 1 + d \cdot 0^2 \cdot 1^2$$

となり, 明らかに成り立つ.

(2) 点の加法逆元

点 $P = (x, y)$ に対して, その逆元 $-P$ は次のように定義される.

$$-P = (x, -y)$$

しかし, 無限遠点の逆元は自分自身であるため,

$$-\mathcal{O} = \mathcal{O} = (0, 1)$$

となる.

(3) スカラー倍

スカラー倍 (kP) で $k = 0$ の場合, 常に無限遠点となる.

$$0 \cdot P = \mathcal{O} = (0, 1)$$

ツイストエドワーズ曲線 E は $\mathcal{O}$ を単位元とする加法群になる.
楕円点の加算, 2 倍算（同一の点の加算）, 無限遠点の加算が同一式で表現されるため, ECDSA のような分岐処理が不要になり, 計算が簡略化される.

3.2 データの変換

EdDSA のアルゴリズム内では, 整数や点をオクテット列に変換するエンコードとその逆変換であるデコードが行われる [**?**].

以下に Ed25519 で使用されるデータの変換について説明する.
なお, b は 8 の倍数とする. また, 16 進数表記の数値は $0x$FF のようにその数値の前に「$0x$」を付けることで表す.

ビット列

8 ビットからなるビット列

$$b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7$$

を**オクテット**と呼ぶ. 厳密には 8 ビット以外を指すこともある「バイト」の代わりに, 必ず 8 ビットのことを指すものとして使われている語である.
ここで, 最下位ビットは b_0, 最上位ビットは b_7 である.

例として, $0x12$ は, 2 進数では 00010010 であり, 8 ビットのビット列で表すと 01001000 となる.

リトルエンディアン形式

リトルエンディアン形式とは, 数値をバイト単位で格納する際に, 最下位バイト（数値の最小の値を持つバイト）を先頭に配置し, 続けてその次に小さいバイトを配置していく方法を指す. これは, 通常の十進法で右端から左へ数字を読むのと逆の順番でデータを並べることになる.

例えば, 32 ビットの $0x12345678$ をリトルエンディアン形式でメモリに格納すると, 次のような順番でバイトが並ぶ：

- 最下位バイト (1 バイト目)：$0x78$
- 2 バイト目：$0x56$
- 3 バイト目：$0x34$
- 最上位バイト (4 バイト目)：$0x12$

このようにして, $0x12345678$ はメモリ上で $0x78, 0x56, 0x34, 0x12$ の順番に格納される.

エンコードとデコード

1. ENC(s)

   整数をオクテット列に変換し, データとして扱いやすくするための関数.

   処理：

   b ビットの整数 s を入力として, s を リトルエンディアン形式にして $\frac{b}{8}$ 個のオクテットに変換して出力する.

2. DEC(t)

   計算で使用できるよう, オクテット列を元の整数に戻す関数.

   処理：

   オクテット列 t を入力として, 整数 s に変換して出力する. つまり, $\mathrm{DEC}(t) = \mathrm{ENC}^{-1}(t) = s$ である.

3. ENCE(A)

   ツイストエドワーズ曲線上の点をオクテット列に変換し, x 座標の符号も付加して効率

的に表現するための関数.

処理：

はじめに符号関数を

$$\mathrm{sign}(a) = \begin{cases} 0 & \text{if } a \geq 0 \\ 1 & \text{if } a < 0 \end{cases}$$

とする. ここで a は整数である.

ツイストエドワーズ曲線上の点 $\mathrm{A}(x, y)$ の y を入力として, $\mathrm{ENC}(y)$ によりオクテット列 $y' = (y'_1, y'_2, ..., y'_{\frac{b}{8}})$ に変換し, $y'_{\frac{b}{8}}$ の最上位ビットに $\mathrm{sign}(x)$ を格納したオクテット列を出力する.

4. $\mathrm{DECE}(t)$

オクテット列を再びツイストエドワーズ曲線上の点 (x, y) に変換する関数. この変換の過程で符号や整合性のチェックを行い, 正しい点を復元する.

処理：

オクテット列 t を入力として, 以下の手順でツイストエドワーズ曲線上の点 (x, y) に変換して出力する.

① t の最終オクテットの最上位ビットを x 座標の符号として取り出し x_0 に格納する. ($x_0 = 0$ または, $x_0 = 1$ とする.)

② t の最終オクテットの最上位ビットを 0 に設定する.

③ $y = \mathrm{DEC}(t)$ を計算し, $0 \leq y < p$ でないならばデコード失敗.

④ 以下の処理を行う.

i. $u = y^2 - 1$, $v = dy^2 + 1$ として $x \equiv uv^3(uv^7)^{\frac{p-5}{8}} \pmod{p}$ を計算する.

ii. $vx^2 \not\equiv \pm u \pmod{p}$ ならばデコード失敗とし, 処理を中断する.

iii. $vx^2 \equiv -u \pmod{p}$ ならば, $x \leftarrow 2^{\frac{p-1}{4}} x$

⑤ $x = 0$ かつ $x_0 = 1$ ならばデコード失敗とし, 処理を中断する.

⑥ x_0 が $x \pmod{2}$ と異なるならば $x \leftarrow p - x$ とする.

⑦ 点 (x, y) を出力する.

3.3 パラメータ

EdDSA のパラメータを表 3.3 にまとめる.

表 3.1 EdDSA のパラメータ

パラメータ	説明
p	法となる素数. EdDSA は $\mathbb{F}_p$ 上の楕円曲線を使用する.
b	$b \geq 10$ かつ $p < 2^{b-1}$ となる正整数. 公開鍵の長さを表す.
H	ハッシュ関数. 2b ビット長のハッシュ値を出力する.
E	EdDSA で使用するエドワーズ曲線. E は $\mathbb{F}_p$ 上の楕円曲線であり, E は以下の式で定義される.
a	E を決定するパラメータ. $\mathbb{F}_p$ 上の平方剰余. すなわち, $x^2 \equiv d \pmod{p}$ となる x が存在する.
d	E を決定するパラメータ. $a \neq d$. 非ゼロの非剰余. すなわち, $x^2 \equiv d \mod p$ となる x が存在しない.
c	E を決定するパラメータ. 2 または 3. 2^c はコファクタと呼ばれる.
L	E を決定するパラメータ. 2^{200} より大きい奇素数で E の位数 $\#E = 2^c l$ となるような数であり, B の位数.
B	E 上のベースポイント. $B \neq (0, 1)$
n	$c \leq n < b$ となる整数.

本研究で使用する Ed25519 のパラメータを表 3.2 にまとめる.

表 3.2: Ed25519 のパラメータ

パラメータ	値, または関数
p	$2^{255} - 19$
b	256
H	SHA-512
E	ツイストエドワーズ曲線　Curve25519
a	-1
d	$-\frac{121665}{121666}$

c	3
L	$2^{252} + 27742317777372353535851937790883648493$
B	(15112221349535400772501151409588531511454012693 04185720604611328394984776220 2, 46316835694926478169428394003475163141307993866256 22 56157830336031652518559 60)
n	254

3.4 デジタル署名アルゴリズム

Ed25519 における 3 つのアルゴリズムの手順を以下に述べる.

なお, ENC, DEC, ENCE, DECE は 2.2 節で述べた関数を使用する. また, H, L, B は 2.3 節で述べた Ed25519 の値, または関数を使用する.

鍵生成

1. 256 バイトのランダムな値 Pk を生成し, 秘密鍵とする.
2. $h = H(Pk) = (h[0] \mid\mid h[1] \mid\mid \cdots \mid\mid h[63])$ を計算する. (ここで $h[n]$ はハッシュ関数の出力 512 ビットをオクテット配列と見なしたときの n オクテット目のビット列を表す.)
3. h を前半部分 $h[0]$ から $h[31]$ と後半部分 $h[32]$ から $h[63]$ に分ける.
4. 前半部分の最初のバイト $(h[0])$ の下位 3 ビットを 0 にクリアする. 最後のバイト $(h[31])$ の最上位ビットを 0 に, 最上位 2 ビット目を 1 に設定したものをリトルエンディアン形式の整数として解釈し, スカラー $s \pmod L$ を生成する. ただし, スカラー値 s は 8 の倍数で, ちょうど 255 ビットとなる.
5. 基準点 B を使って $A = sB$ を計算し, $ENCE(A)$ を公開鍵とする.

署名生成

1. 秘密鍵 Pk を使って, ハッシュ値 $h = H(Pk)$ を計算する.
2. h の後半部分 $h[32]$ から $h[63]$ を使って,

$$r \equiv H(h[32] \,||\, \cdots \,||\, h[63] \,||\, M) \mod L$$

を計算し, デコードする.

3. $R = rB$ を計算し, エンコードする.
4. $k = H(R \,||\, A \,||\, M)$ を計算する.
5. 鍵生成の際 s を用いて $S \equiv r + ks \pmod{L}$ を計算し, エンコードする.
6. (R, S) を署名とする.

署名検証

1. メッセージ M と署名 (R, S) を受け取る.
2. $R' = DECE(R)$ として, R をデコードする.
3. $S' = DEC(S)$ として, S をデコードする.
4. $A' = DECE(A)$ として, A をデコードする.
5. 2〜4 でデコードに失敗した場合や $0 \leq S < L$ の範囲外である場合, 署名を棄却する.
6. $k' = H(R' \,||\, A' \,||\, M)$ を計算する.
7. $8S'B = 8R' + 8k'A'$ が成り立てば, 署名は有効である. そうでなければ, 署名は無効である.

3.5 ECDSA と EdDSA の比較

EdDSA は ECDSA と比べ, 以下の点でセキュリティと処理時間が向上している.

セキュリティ

決定論的な署名生成

Ed25519 は, 署名ごとにランダムな値を生成する代わりに, メッセージと秘密鍵をハッシュすることでノンス（署名に使用するランダムな数値）を決定論的に生成する. この方法により, 乱数生成の失敗による秘密鍵の漏洩リスクを回避している. 一方, ECDSA では, 乱数が予測可能または重複した場合, 秘密鍵が簡単に推測されるリスクがあり, これがセキュリティの大きな弱点となる.

処理時間

乱数生成の回避

EdDSA では乱数を鍵生成でのみ使用している．この乱数は秘密鍵に直接影響を与えるため，暗号的に安全である必要がある．一方で，署名生成，署名検証では乱数を使用しない．一般的に，暗号的に安全な乱数は生成の時間がランダムで処理コストが高い．EdDSA では署名生成，署名検証を一定時間で行い，乱数を使用するアルゴリズムに比べて高速に処理することができる．

乗法逆元の不要

Ed25519 では拡張した射影座標系（拡張ツイストエドワーズ座標）で計算することが RFC8032 で推奨されている． この座標系の特性により, 処理コストの高いスカラー値の乗法逆元の処理が不要となる. したがって, 処理が単純化され, 高速化しやすい.

第 4 章　提案手法

第 5 章　シミュレーション環境

この章では, 本研究で行ったシミュレーション環境について述べる. シミュレーションのパラメータは表 5.1 の通りである.

表 5.1: シミュレーションパラメータ

ホスト OS	Windows 10
ゲスト OS	Ubuntu 16.02_LTS
シミュレーションツール	ns-3.26
通信規格	IEEE 802.11p
通信プロトコル	UDP
パケットサイズ	1024 [byte]
パケット送信間隔	1.0 [s]
送信電力	17.026 [dBm]
電力検出閾値	-96.0 [dBm]
電波伝搬減衰モデル	対数距離電波伝搬減衰モデル
遅延モデル	定常速度伝搬モデル
電波伝搬範囲	約 300 [m]
ノード数	74(実験 1, 2), 37/74/112/148/185(実験 3)
シミュレーション時間	300 [s]
ルーティングプロトコル	GPSR
デジタル署名	DSA, ECDSA, EdDSA

ns-3

本研究では, シミュレーションツールとして **network simulator-3(ns-3)** [?] 使用した. ns-3 は, 離散事象ネットワークシミュレータであり, 有線および無線通信プロトコルを含む多様なネットワークのシミュレーションが可能なオープンソースソフトウェアである. ns-3 のシステムは大きく分けて, シミュレーションの実行を行う **ns-3 core** と, 実験の定義を行う **simulation scenario** に分かれている. ユーザーは, 自身のシミュレーションの要求に自身のシミュレーションの要件に応じて不足している部分を適宜実装し, シミュレーションを実行

する．開発言語は C++と Python をサポートしているが, ns-3 core は C++でのみ改変することができるため, 本研究では C++でコーディングした.

ns-3 では様々なコンテナと呼ばれるノードやネットワーク要素を効率的に管理するためのデータ構造が存在し, それらを組み合わせてプログラムを作成していく．一般的には次の 4 つの主要なコンテナが使用される.

- NodeContainer
  ノードを管理するためのコンテナであり, コンピュータやルータなど, 扱うデバイスが何であるかを示す.
- DeviceContainer
  通信デバイスを管理するためのコンテナであり, ノードがどのような通信機能をもつかを指定する.
- InterfaceContainer
  IP インターフェースを管理するためのコンテナであり, IP アドレスやサブネットマスクなどの情報を保持する.
- ApplicationContainer
  アプリケーションレイヤのプログラムやサービスを管理するためのコンテナであり, HTTP サーバや UDP アプリケーションなどが用意されている.

ns-3 は Linux 環境での動作を前提に開発されている． そのため, 本研究では仮想環境 VMware に Ubuntu をインストールし, その上で ns-3 を動作させた． 現時点での ns-3 の最新バージョンは 2024 年 10 月 9 日リリースの ns-3.43 となっているが, 本研究では, 先行研究 [**?**] と互換性のある ns-3.26 を使用した.

通信規格 IEEE802.11p

また, 本研究では通信規格として IEEE802.11p を想定した． **IEEE802.11p** とは, 車車間通信 (V2V) や路車間通信 (V2I) を可能にするために設計された無線通信規格であり, IEEE が定める 802.11 シリーズ (Wi-Fi 規格) の一部である． この規格は, 高度道路交通システム (Intelligent Transportation System, ITS) の通信要件を満たすため, 主に交通安全や効率化を目的としたアプリケーションに使用される.

この規格の特徴を以下に示す.

- **OFDM(直行周波数分割多重方式)**
  IEEE802.11p は, IEEE802.11a をもとに設計されており, データ送信に OFDM を採用し

ている．**OFDM** とは, サブキャリア (OFDM で 1 つのチャネル (10MHz) 内に細かく分けた周波数成分) を直交する形で並べて, つまり, 互いに干渉しないように送信することにより, 周波数帯域を効率的に利用できる多重化技術である． また, この方式は信号の反射による複数経路からの干渉 (マルチパス干渉) に対して強い耐性を持ち, 高速移動環境下でも安定した通信を可能にする． さらに, 通信速度を表すデータレートは 6Mbps から 27Mbps の範囲で柔軟に設定できるため, さまざまな通信条件に適応可能である.

- **周波数帯域**

  IEEE802.11p では, 通信に使用される周波数帯域として 5.850GHz 5.925GHz(5.9GHz 帯) が ITS 専用として割り当てられている．チャネル構成としては, 標準の Wi-Fi で用いられる 20MHz のチャネル幅を半減し, 10MHz のチャネル幅を採用している．この 10MHz 単位の幅で 7 つのチャネルが定義されており, 各チャネルは 10MHz 間隔で配置される．この中でも, 安全通信用として制御チャネル (CCH) とサービスチャネル (SCH) という, 2 つの特別なチャネルが確保されている．CCH は事故発生通知, 赤信号の警告, 緊急車両の接近通知などの緊急通信, SCH は道路状況, 渋滞情報, 駐車場の空き情報などの便利な情報を伝えるための通信をするチャネルとなっており, これらを時間で切り替えながら通信を行っている．この設計により, リアルタイム性が求められる交通安全アプリケーションに適した高い信頼性と効率性を実現している.

対数距離電波伝搬減衰モデル

ns-3 では通信環境をシミュレーションする際に, 電波の伝搬特性を表すために, 様々な伝搬モデルが用意されている． 本研究では, 都市, 郊外, 屋内といった様々な環境に適用できる対数距離電波伝搬減衰モデルを使用した.

対数距離電波伝搬減衰モデルの定義を式 5.1 に示す． ここで, d は送信機と受信機間の実際の距離, d_0 は参照距離 [m], L は距離 d での伝搬損失 (dB), L_0 は参照距離 d_0 での伝搬損失, n は環境依存のパケットロス指数である．参照距離とは, 電波の反射や回折などによる減衰を考慮して決められる値である.

$$L = L_0 + 10n \log_{10} \left(\frac{d}{d_0} \right) \tag{5.1}$$

定常速度伝搬モデル

定常速度伝搬モデル (Constant Speed Propagation Delay Model) は, このモデルは電波が空間を伝搬する速度が一定であるという仮定に基づいており, 伝搬遅延 Δt は, 送信ノードと受

信ノード間の距離 d と電波の伝搬速度 v によって以下の式で定義される.

$$\Delta t = \frac{d}{v}$$

移動モデル　SUMO

まだ書いてない
デジタル署名

本研究では, デジタル署名に 1.3 節で述べた DSA と ECDSA に加え, 2 章で解説した EdDSA(Ed25519) を使用してシミュレーションを行い, デジタル署名方式による比較をした.

DSA のセキュリティパラメータは以下の通りである.

$$(L, N) = (2048, 256)$$

ECDSA では, 楕円曲線に scep256kl を使用した. その式とグラフ (図 4.4) は以下の通りである.

$$y^2 = x^3 + 7$$

図 4.4

EdDSA のパラメータは 2.3 節で述べたものを使用した.

第 6 章　実験

本研究では, 第 4 章で述べたシミュレーション環境を用いて 3 種類のシミュレーション実験を行った. いずれの実験でも, 以下の 4 パターンを調べた.

- 認証機構を用いない場合
- DSA を用いて認証機構を追加した場合
- ECDSA を用いて認証機構を追加した場合
- EdDSA を用いて認証機構を追加した場合

シミュレーション結果を評価するために, 以下の 4 つの評価基準を用いる.

1. **スループット (TP)**

   スループットとは, 単位時間あたりに正常に転送されたデータ量のことである. 以下に示す式で定義される. ここで, $AllTxBytes$ は合計転送バイト数, $TxTimes$ は転送にかかった時間である.

$$TP = \frac{AllTxBytes \times 8}{TxTimes \times 1000}[\text{kbps}]$$

2. **遅延時間 (DT)**

   遅延時間とは, パケットが送信されてから受信されるまでの時間のことである. 以下に示す式で定義される. ここで, $TxPacketTime$ は送信ノードがパケットを送信した時間, $RxPacketTime$ は宛先ノードがパケットを受信した時間である.

$$DT = RxPacketTime - TxPacketTime[\text{ms}]$$

3. **パケット配送率 (PDR)**

   パケット配送率とは, 送信されたデータ量のうち損失せずに受信されたデータ量の割合のことである. 以下に示す式で定義される. ここで, $AllTxPackets$ は合計送信パケット数, $AllRxPackets$ は合計受信バイト数である.

$$PDR = \frac{AllRxPackets}{AllTxPackets} \times 100[\%]$$

4. **オーバーヘッドサイズ (OH)**

   オーバーヘッドサイズとは, データの送受信に付随して発生する余分なコストのこと

であり, ここではルーティングに使用される通信データ量を指す. 以下に示す式で定義される. ここで, $AllTxKBytes$ は Hello パケットを含めた全ノードの合計送信キロバイト数, $TxKBytes$ はデータパケットの合計送信キロバイト数である.

$$OH = AllTxKBytes - TxKBytes[\text{KB}]$$

3 種類の実験の概要を述べる.

実験 1 では, 不正ノードが存在する環境で 4 パターンのシミュレーションを行い, 認証機構の有効性を調査する. 具体的には, パケット配送率 (PDR) とスループットを測定し, EdDSA によってセキュリティがどの程度維持されているかを評価する. 実験 2 では, 実験 1 の結果をもとに, 認証機構が正しく機能していることを確認したうえで, 署名方式の違いが通信品質にどう影響するのかについて調査する. 具体的には, 平均遅延時間と平均パケット配送率を測定し, オーバーヘッドサイズを測定し, EdDSA が他の認証方式と比べてどのような特徴を持つのかを評価する. 実験 3 では, シミュレーション実行時間と署名生成, 検証にかかる時間を計測し, EdDSA の処理効率について評価する.

6.1 実験 1

実験 1 では, 認証機構が正しく機能していることを確認するための実験を行った. 適度に影響が出るよう, 3 章で述べた 2 種類の不正ノードを 3 個ずつ用意し, 全ノードの約 8% (6 個のノード) を不正ノードに設定した. また, 結果のばらつきを抑えつつ, 統計的に有意な評価を可能にするために, そのような環境で 250 回シミュレーションを行い, パケット配送率 (PDR) とスループットを調べた.

実験の結果は以下の通りである.

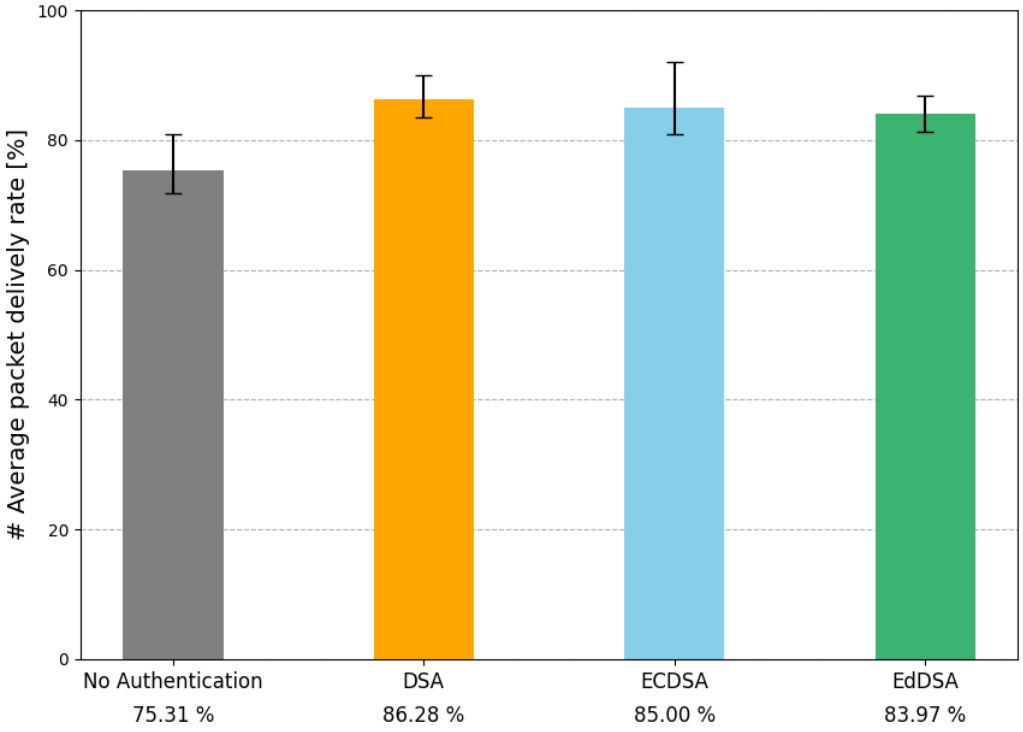

図 6.1　不正ノードが存在する環境でのパケット配送率

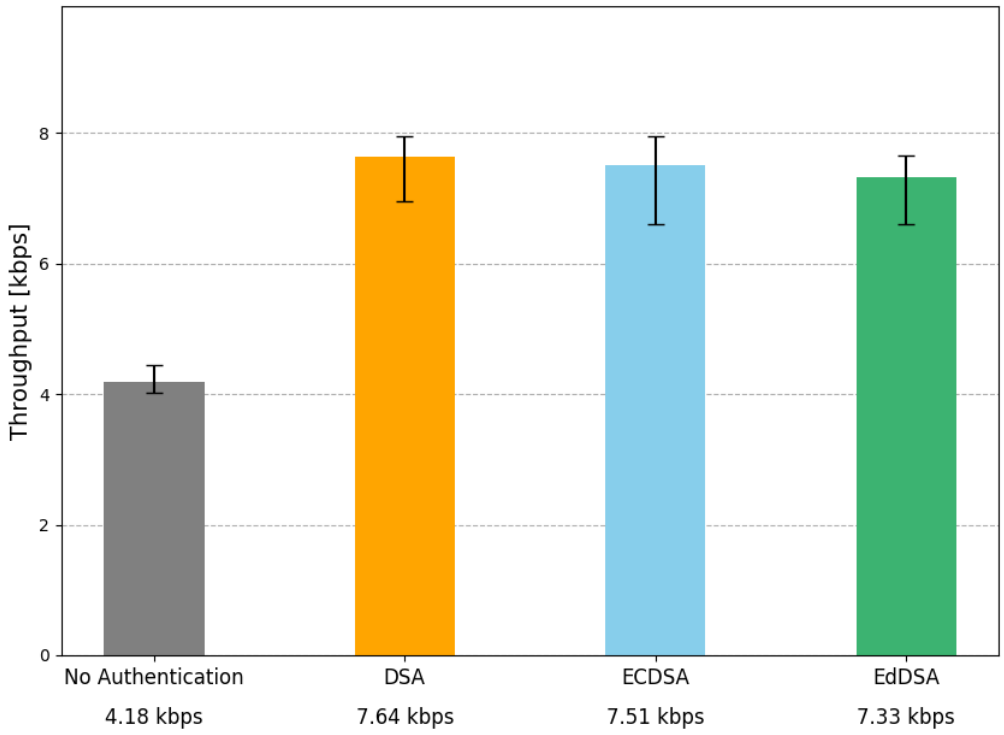

図 6.2　不正ノードが存在する環境でのスループット

図 6.1 は, 実験 1 におけるシミュレーションパターンごとのパケット配送率を示している. 認証機構なしの場合に 48.2 %であるのに対し, DSA では 88.15 %, ECDSA では 86.62 %, EdDSA では 84.48 %と, 認証機構を追加したことでパケット配送率が約 30 %向上した.

図 6.2 は, 実験 1 におけるシミュレーションパターンごとのスループットを示している. 認証機構なしの場合に 4.18kbps であるのに対し, DSA では 7.64kbps, ECDSA では 7.51kbps,EdDSA では 7.33kbps と, 認証機構を追加することでパケット配送率同様, スループットも向上した.

これらの結果は, 認証機構の追加により不正ノードが排除されたことで, 経路選択を行う際に正当なノードのみを選択しており, データの窃取 (転送中止) を回避できたことを示している. しかし, EdDSA の結果を DSA と ECDSA の結果と比較するとほとんど差がないため, 3 つの署名方式が不正ノードを排除することにおいて同等の性能をもつと考えられる.

6.2 実験 2

実験 2 は認証機構の追加が通信にどれだけの負荷を与えるのかを調査するための実験を行った. 不正ノードの存在しない環境で, 250 回シミュレーションを行い, 遅延時間, パケット配送率, オーバーヘッドサイズを調べた.

実験の結果は以下の通りである.

表 6.1: 遅延時間の中央値

プロトコル	中央値 [ms]	最頻値 [ms]
No Authentication	6.88	$[6.0, 7.0)$
DSA	6.13	$[2.0, 3.0)$
ECDSA	6.20	$[2.0, 3.0)$
Ed25519	6.42	$[6.0, 7.0)$

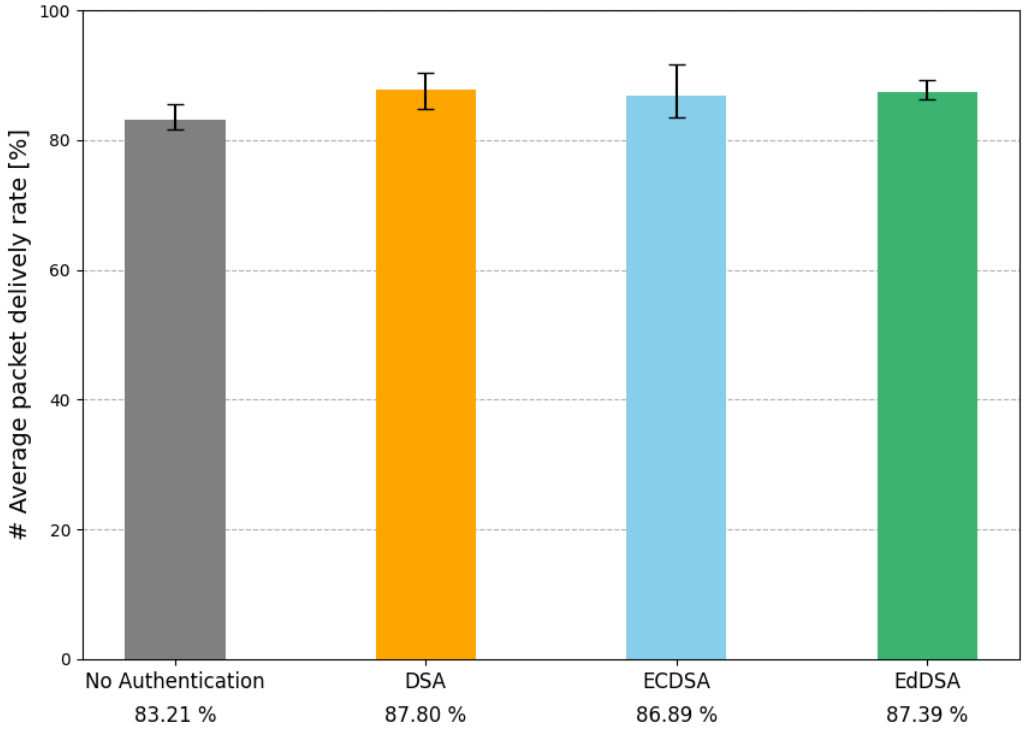

図 6.3　不正ノードが存在しない環境でのパケット配送率

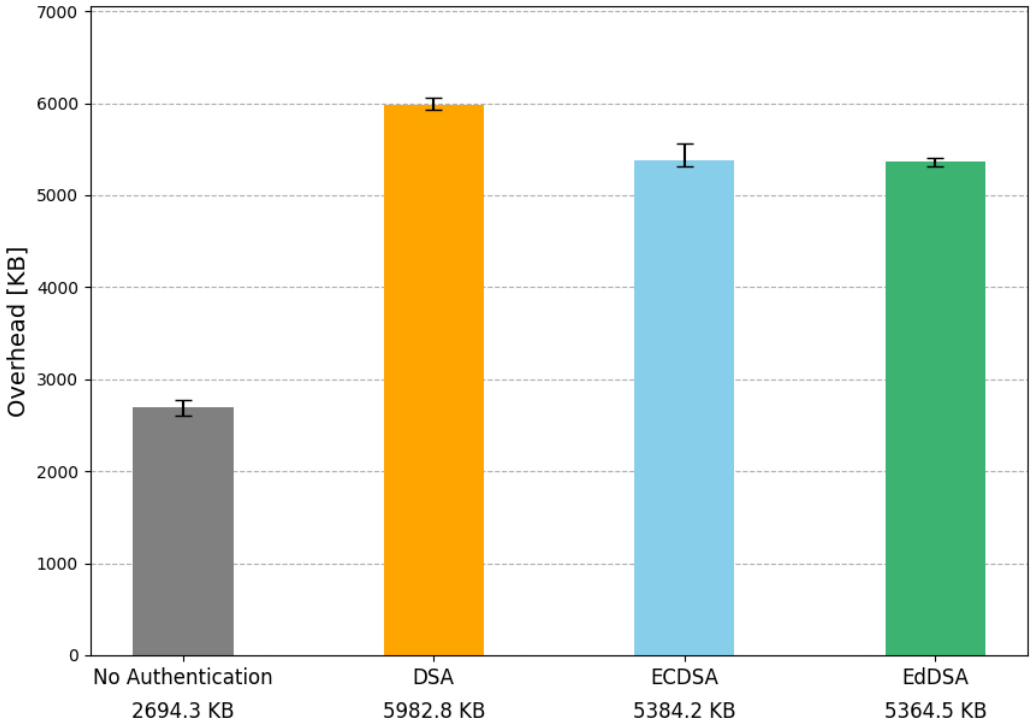

図 6.4　オーバーヘッドサイズ

表 6.1 は, 実験 2 におけるシミュレーションパターンごとの遅延時間の中央値と最頻値を示している. 中央値について, 認証機構なしでは 6.88ms, DSA では 6.13ms, ECDSA では 6.2ms, EdDSA では 6.42ms であった. 最頻値について, 認証機構なしと Ed25519 では 6.0ms から 7.0ms, DSA と ECDSA では 2.0ms から 3.0ms の範囲が最も多かった. この結果は, 遅延時間は経路選択のタイミングによって左右されるため, 若干の差異が出てしまうが, 認証機構の有無が遅延時間に影響を与えなかったことを示唆している.

図 6.3 は, 実験 2 におけるシミュレーションパターンごとのパケット配送率を示している. 認証機構なしでは 83.21 %, DSA では 87.8 %, ECDSA では 86.89 %, EdDSA では 87.39 %となった. この結果から, 認証機構の有無はパケット配送率に影響を与えなかったと考えられる.

図 6.4 は, 実験 2 におけるシミュレーションパターンごとのオーバーヘッドサイズを示している. 認証機構なしでは 2694.3KB であったのに対し, DSA では 5982.8KB, ECDSA では 5384.2KB, EdDSA では 5364.5KB と, 認証機構の追加によりオーバーヘッドサイズが大幅に増加した. これは, Hello パケットのデータに署名が付与されていることが原因である. また, EdDSA の結果を DSA と ECDSA の結果と比較すると, EdDSA と DSA では大きな差があった

のに対し，EdDSA と ECDSA ではほとんど差がなかった． EdDSA が DSA よりも鍵長が短いが，ECDSA とは変わらないことからこのような結果になったと考えられる． なお，本研究で使用したパラメータによるそれぞれの鍵長は，DSA で 2048 ビット，ECDSA と EdDSA で 256 ビットである．

6.3 実験 3

実験 3 では，認証機構の処理能力 (計算効率) を評価するための実験を行った． 不正ノードの存在しない環境でノード数を 37, 74, 112, 148, 185 に設定して 250 回ずつシミュレーションを行い，それぞれの実行にかかった時間を調べた．

実験の結果は以下の通りである．

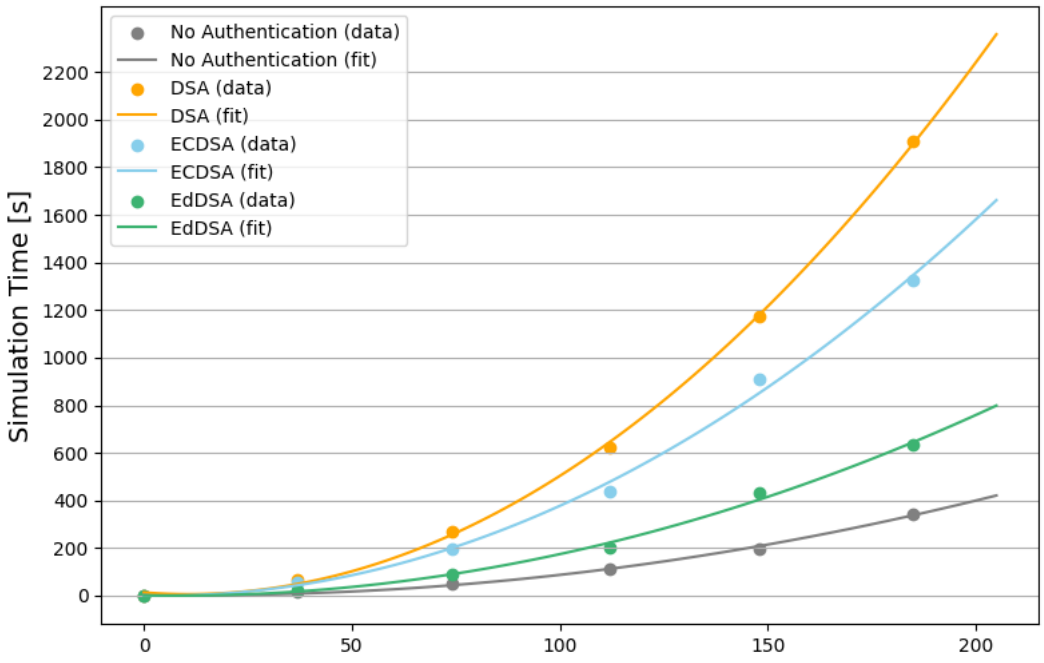

図 6.5 ノード数によるシミュレーション実行時間の変化

表 6.2: ノード数によるシミュレーション実行時間

Number of nodes	No Authentication $[s]$	DSA $[s]$	ECDSA $[s]$	Ed25519 $[s]$
37	14.1802	69.9882	55.0108	24.4069
74	50.5984	267.433	194.408	89.2728
112	110.405	625.504	436.926	200.901
148	196.971	1172.57	910.373	431.346
185	345.059	1908.7	1324.55	635.155

表 6.3: 1 回の署名作成と署名検証にかかった時間

プロトコル	署名作成時間 $[ms]$	署名検証時間 $[ms]$
ECDSA	0.371	0.336
Ed25519	0.031	0.097

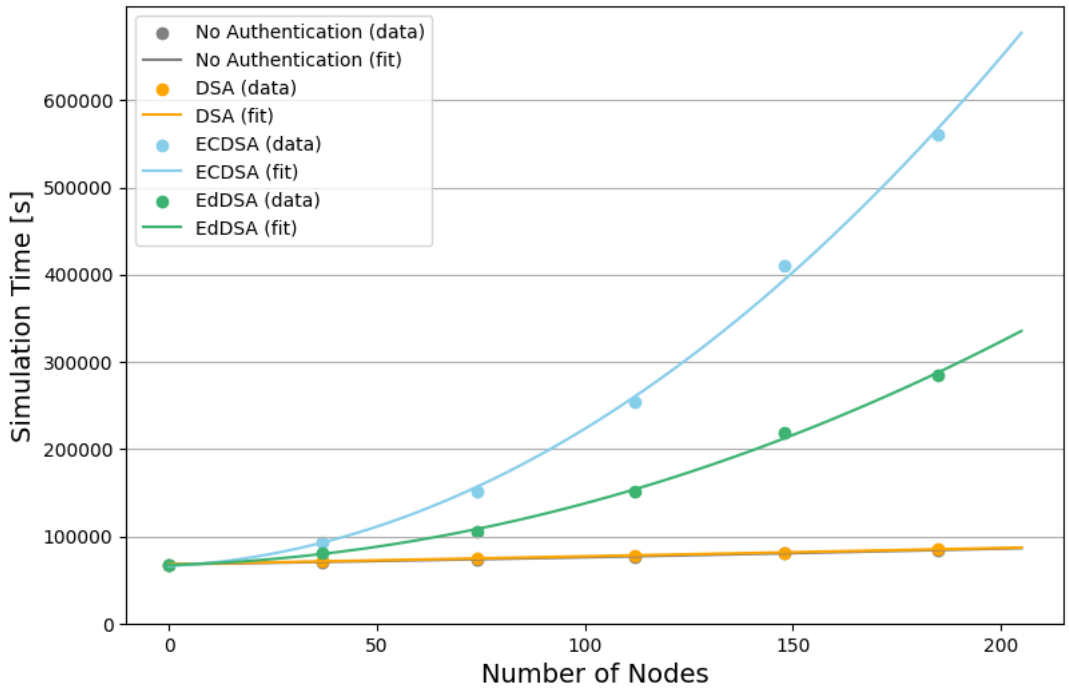

図 6.6　ノード数によるメモリ使用量

表 6.4: ノード数によるメモリ使用量

Number of nodes	No Authentication [KB]	DSA [KB]	ECDSA [KB]	Ed25519 [KB]
37	70939.4	72023.3	93749.3	80647.1
74	74089.7	74989.3	151853	106442
112	77070.1	78134.7	254247	151401
148	80749.9	81752.5	409965	219102
185	84548.1	85541.4	560753	285191

図 6.5 はシミュレーションパターンごとのノード数による実行時間の変化を近似してグラフ化したものを, 表 6.2 はその具体的な値を示したものである. 表 6.3 には ECDSA と EdDSA において, 1 回の署名作成と署名検証にかかった時間を示した. 図 6.6 は, ノード数によるメモリ使用量の変化を近似してグラフ化したものを, 表 6.4 はその具体的な値を示したものである.

ECDSA と EdDSA に注目して結果を評価する. 図 6.5, 表 6.3 より, EdDSA, ECDSA の順番で実行時間が短く, その差はノード数が増加するほど大きくなっている. 言い換えると, ECDSA の実行時間の増加率が大きいのに対し, EdDSA は比較的緩やかである. また, 表 6.3 より, EdDSA が ECDSA に比べて署名生成, 署名検証にかかる時間が短いことから, EdDSA は ECDSA よりも計算効率が良いということが確かめられた. さらに, 図 6.6 より, EdDSA のメモリの使用量が ECDSA よりも少なく, 実行時間と同様に ECDSA の実行時間の増加率が大きいのに対し, EdDSA は比較的緩やかである. これらの結果から, EdDSA は ECDSA と比較してネットワーク全体の計算コストの増加を抑えつつ, より少ないリソースで効率的に動作することが可能であり, 特に大規模なネットワーク環境において優れたスケーラビリティを有すると考えられる.

第 7 章　EdDSA に関する実装評価まとめ

この章では, 2 章で説明した EdDSA の性能と 5 章で示した実験結果から, EdDSA の実装評価をする. はじめに, 5 章で示した実験結果の評価を他の認証方式と比較しながらまとめ, その後に 2 章の内容を含めながら ECDSA との総合的な比較評価を行う. 最後に, 本研究で行った実験結果から想定される EdDSA の利用シーンについて考察する.

実験結果のまとめ

1. 実験 1

   EdDSA を用いたことで, 不正ノードを排除することができたが, 他の認証方式との差は見られなかった.
2. 実験 2

   平均遅延と平均パケット配送率において, EdDSA は他の認証方式との差は見られなかったが, それぞれの鍵長の違いからオーバーヘッドサイズが DSA よりも小さく, ECDSA と同程度であった.
3. 実験 3

   EdDSA は他の認証方式よりも署名の生成と検証にかかる時間が短いことから, 処理能力 (計算効率) が高いことがわかった.

EdDSA と ECDSA の比較評価

上記の実験結果のまとめから, EdDSA は ECDSA よりも計算効率が高いことがわかった. これは, 第 2 章で述べた EdDSA の設計上の特長である高速な処理能力が実験環境においても十分に発揮されたことを示している. また, 第 2 章で述べたセキュリティの堅牢性から, EdDSA は秘密鍵を特定しようとする攻撃者が存在する環境で ECDSA よりも高いセキュリティを提供できる. よって, V2V アドホックネットワークにおいて, EdDSA は ECDSA よりも安全かつ効率的な認証方式として利用できるといえる.

EdDSA の利用シーンについての考察

シミュレーション実験から EdDSA は他の認証方式, 特に ECDSA と比べても通信品質が向上, または同等であり, スケーラビリティも優れていることがわかった. しかし, その結果から, 本研究の実験環境では, 署名の生成と検証の時間がデータパケットと Hello パケットの送信間隔 (1 秒) に比べて非常に短い. そのため, 署名に関する処理が通信全体に及ぼす影響が大

変小さく, EdDSA の特徴を最大限発揮できていないということが予想される. したがって, 次のような環境であれば署名の生成と検証の回数がより多くなり, EdDSA の処理能力とスケーラビリティを最大限発揮できると考える.

- ノード数が非常に多い
- データパケットと Hello パケットの送信間隔が非常に短い
- 使えるリソースが限られている

さらに, 次のような環境であれば EdDSA のセキュリティの堅牢さを発揮すると考えられる.

- 秘密鍵を特定しようとする攻撃者が存在する

おわりに

本論文は 6 章で構成される．第 1 章では, VANET やデジタル署名について説明した．第 2 章では, 本研究の対象である EdDSA(Ed25519) の特徴やアルゴリズムについて解説した．第 3 章では, セキュアにルーティングを行うためにの認証機構の導入方法について解説した．第 4 章では, シミュレーション環境について述べ, 第 5 章でシミュレーションによる実験を行い, その結果と評価を示した．第 6 章では, 第 2 章と第 5 章の内容を踏まえ, EdDSA の実装評価について議論した．

本研究では, EdDSA の高い計算効率および堅牢なセキュリティ性能が, 認証機構を追加した GPSR による V2V 通信にどのような影響を与えるかを検証した．その結果, EdDSA を使用した場合, 署名生成および検証にかかる処理時間が他の署名方式 (ECDSA, DSA) よりも短縮され, ネットワーク全体の計算負荷が軽減されることを明らかにした．また, EdDSA の性能を最大限活かすための利用シーンの考察すると, EdDSA は V2V アドホックネットワークや IoT が発展に伴い, 今後ますます重要な役割を果たすと考えられる．

謝辞

参考文献