Forward Email: Technical Whitepaper

The only 100% open-source, encrypted, privacy-focused, transparent, and quantum-resistant email service

Nicholas Baugh, Founder
Forward Email LLC
nick@forwardemail.net
https://forwardemail.net

First Published: April 11, 2025

Last Revised: April 11, 2025

Abstract

In an era where digital privacy is increasingly compromised, Forward Email stands as a beacon of transparency and security in email infrastructure. As the only 100% open-source, encrypted, privacy-focused, transparent, and quantum-resistant email service, we have fundamentally reimagined what users should expect from their email provider.

Since our founding in 2017, Forward Email has grown to power email for over 500,000 domains – including industry leaders such as Canonical, Netflix, The Linux Foundation, The PHP Foundation, Fox News Radio, Disney Ad Sales, jQuery, LineageOS, Ubuntu, Kubuntu, Lubuntu, The University of Cambridge, The University of Maryland, The University of Washington, Tufts University, Swarthmore College, Government of South Australia, Government of Dominican Republic, Fly.io, RCD Hotels, International Correspondence Chess Federation, and notable developers like Isaac Z. Schlueter (npm creator) and David Heinemeier Hansson (Ruby on Rails creator). This remarkable adoption stems from our steadfast commitment to privacy-by-design principles and complete technical transparency.

This whitepaper provides extensive detail and technical insight into our history, threat modeling, systems and security architecture, and future roadmap. Through comprehensive analysis of our implementation choices, cryptographic approaches, and security protocols, we demonstrate how Forward Email achieves unparalleled privacy protection while maintaining full auditability through open-source development.

By combining quantum-resistant encryption, zero-knowledge operations, and a distributed architecture with our steadfast refusal to collect user metadata, Forward Email represents not just an email service, but a fundamental shift in how privacy-critical infrastructure can and should be built in the digital age.

Dedicated to my loving dog and best friend, Jack, whose unwavering companionship supported the creation of this service.

Contents

1 Introduction

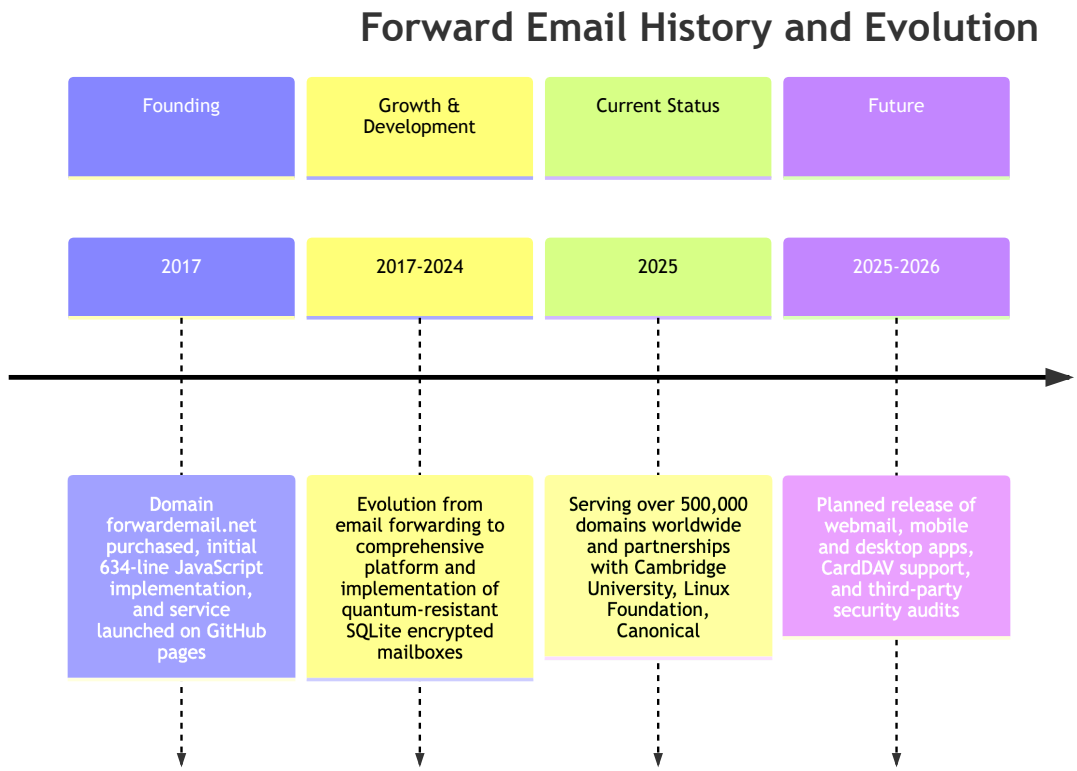
Forward Email stands at the forefront of email privacy and security, offering a unique approach to email forwarding and hosting that prioritizes transparency, security, and user control. Founded in 2017, Forward Email has grown to serve over 500,000 domains worldwide, including notable organizations such as Canonical (Ubuntu), Netflix, The Linux Foundation, The PHP Foundation, Fox News Radio, Disney Ad Sales, jQuery, LineageOS, The University of Maryland, The University of Washington, Tufts University, Swarthmore College, Government of South Australia, Government of Dominican Republic, RCD Hotels, International Correspondence Chess Federation, Isaac Z. Schlueter (npm), and David Heinemeier Hansson (Ruby on Rails)[1].

1.1 History and Evolution

In early 2017, Nicholas Baugh (the founder of Forward Email) was in search of a cost-effective and simple solution for enabling email on domain names for his side-projects[1]. After researching available options, Baugh began coding his own solution and purchased the domain `forwardemail.net` on October 2, 2017[2].

One month later, on November 5, 2017, Baugh created a 634-line JavaScript file using Node.js to forward emails for any custom domain name. This initial implementation was published as open-source to GitHub[2] and the service was launched using GitHub Pages, as evidenced by early web archives[3].

[1] Paul Graham's essay on side projects: https://paulgraham.com/own.html
[2] Initial GitHub commit: https://github.com/forwardemail/free-email-forwarding/commit/5b1fda800972db191b9a2b9dbeeb333fc71
[3] Early web archive of Forward Email: https://web.archive.org/web/20171119040111/http://forwardemail.net/#/



In the early versions of Forward Email, the service only provided email forwarding and was purely DNS-based. There was no account registration or sign-up process—it was simply a README file written in Markdown with instructions. Users could (and still can) set up email forwarding on their custom domain by configuring MX records to point to `mx1.forwardemail.net` and `mx2.forwardemail.net`, and adding a TXT record with `forward-email=user@gmail.com`. This simple approach allowed all emails sent to any address at the user's domain to be forwarded to their specified email address.

The simplicity and effectiveness of this solution attracted attention from prominent developers, including David Heinemeier Hansson (creator of Ruby on Rails), who continues to use Forward Email[4] on his domain `dhh.dk` to this day.

Since its inception, Forward Email has maintained a steadfast commitment to privacy and security principles:

- **100% Open-Source Philosophy**: Unlike competitors who only open-source their frontends while keeping backends closed[3], Forward Email has made its entire codebase—both frontend and backend—available for public scrutiny on GitHub.
- **Privacy-First Design**: From day one, Forward Email implemented a unique in-memory processing approach that avoids writing emails to disk, setting it apart from conventional email services that store messages in databases or file systems.

[4]David Heinemeier Hansson's tweet about using Forward Email: https://x.com/dhh/status/1331183379686055936

- **Continuous Innovation**: The service has evolved from a simple email forwarding solution to a comprehensive email platform with features like encrypted mailboxes, quantum-resistant encryption, and support for standard protocols including SMTP, IMAP, POP3, and CalDAV.

1.2 Mission and Values

Forward Email's mission extends beyond providing email services—it aims to transform how the industry approaches email privacy and security. The company's core values include:

1. **Transparency**: Making all code open-source and available for inspection, ensuring users can verify privacy claims rather than simply trusting marketing statements[4].
2. **User Control**: Empowering users with options, including the ability to self-host the entire platform if desired, as detailed in the self-hosted solution documentation[5].
3. **Privacy Protection**: Implementing technical measures that ensure user data remains private by design, not just by policy, through features like in-memory processing and encrypted storage.
4. **Security Innovation**: Pioneering advanced security measures like quantum-resistant encryption and individually encrypted SQLite mailboxes, as evidenced in the codebase.
5. **Accessibility**: Maintaining affordable pricing that doesn't scale with user count, making privacy-focused email accessible to all.

1.3 Notable Achievements

Forward Email has achieved several significant milestones that demonstrate its impact on the email service landscape:

- Implementing quantum-resistant encryption through the use of ChaCha20-Poly1305 cipher for mailbox encryption[5] [6]
- Developing a unique approach to email storage using individually encrypted SQLite mailboxes[7] with optimized PRAGMA settings[8]
- Growing to serve over 500,000 domains without relying on third-party services like Amazon SES[9]
- Establishing partnerships with prestigious institutions including The University of Maryland[10], The Linux Foundation[11], and Canonical (Ubuntu)[12]
- Achieving perfect scores on security tests from Internet.nl, SSL Labs, Mozilla Observatory, and Hardenize[13]
- Creating and maintaining open-source npm packages that have collectively reached over 1 billion downloads, significantly contributing to the JavaScript ecosystem[14]

[5]Forward Email's self-hosting documentation: https://forwardemail.net/blog/docs/self-hosted-solution

1.3.1 Open-Source JavaScript Contributions

Forward Email's commitment to open source extends beyond its email service. The team has developed and maintains several critical npm packages that have become essential components of the JavaScript ecosystem:

- **Bree**: A modern job scheduler with over 3,100 GitHub stars that uses Node.js worker threads for better performance and reliability. It doesn't force users to use any specific database, allowing them to decide what databases to integrate for job locking. Users can choose to use Redis, MongoDB, or other databases only for jobs that require them, making jobs more lightweight in memory consumption and reducing potential points of failure[15]
- **Cabin**: A structured logging system with nearly 900 GitHub stars and over 100,000 weekly downloads, providing powerful logging capabilities for modern applications[16]
- **Spam Scanner**: Advanced email spam detection tools, including the widely-used `url-regex-safe` package with over 1.2 million downloads in two months, which fixes critical security issues in URL detection regular expressions[17]
- **Tangerine**: A DNS over HTTPS implementation that serves as a drop-in replacement for Node.js DNS with built-in retries, timeouts, smart server rotation, and caching support[18]

These contributions demonstrate Forward Email's technical expertise and commitment to improving the broader open-source ecosystem, while also enhancing the reliability and security of their own email platform.

1.4 Current Position in the Market

As of 2025, Forward Email occupies a unique position in the email service market. While competitors like Proton Mail and Tutanota have gained recognition for their privacy-focused approaches, Forward Email differentiates itself through:

- Complete open-source transparency (both frontend and backend), unlike Proton Mail's closed-source backend[3] and Tutanota's closed-source implementation[19]
- In-memory email processing that avoids disk storage, enhancing privacy and security
- Individually encrypted SQLite mailboxes[7] rather than shared relational databases
- Quantum-resistant encryption implementation using ChaCha20-Poly1305[6]
- Support for standard email protocols (SMTP[20], IMAP[21], POP3[22]) without requiring proprietary bridges or apps
- A pricing model that doesn't charge per user, making it more accessible for organizations

These differentiators have allowed Forward Email to carve out a distinct niche in the market, appealing particularly to privacy-conscious individuals, organizations with compliance requirements, and technical users who value transparency and control.

The following sections of this whitepaper will explore Forward Email's approach to threat modeling, systems architecture, security implementation, DNS infrastructure, and future roadmap, providing a comprehensive overview of how the service achieves its privacy and security goals

through technical innovation and principled design.

2 Threat Modeling

Forward Email's approach to security is built on a comprehensive threat modeling framework that identifies potential vulnerabilities, assesses risks, and implements appropriate countermeasures. This section outlines the threat model that guides Forward Email's security architecture and practices, with actual code examples demonstrating how these security principles are implemented.

2.1 Threat Modeling Approach

Forward Email employs a systematic approach to threat modeling that combines elements of multiple industry-standard frameworks. Our primary methodology is based on STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege)[23], supplemented with concepts from LINDDUN for privacy-focused threat analysis and PASTA (Process for Attack Simulation and Threat Analysis)[24] for risk-centric assessment[25]. This comprehensive approach allows for both structured analysis of potential threats and creative exploration of attack vectors.

The threat modeling process follows these key steps:

1. **System Decomposition**: Breaking down the email system into its component parts and data flows
2. **Threat Identification**: Identifying potential threats to each component using the STRIDE framework
3. **Privacy Analysis**: Evaluating privacy implications using LINDDUN principles
4. **Risk Assessment**: Evaluating the likelihood and impact of each identified threat
5. **Mitigation Strategy**: Developing and implementing countermeasures for each significant threat
6. **Continuous Validation**: Regularly reviewing and updating the threat model as the system evolves

2.2 Adversary Models

Forward Email's threat modeling considers several categories of potential adversaries, drawing inspiration from Mullvad VPN's security approach[26]:

2.2.1 Nation-State Actors

- **Capabilities**: Advanced persistent threats, sophisticated surveillance infrastructure, significant computational resources including potential quantum computing capabilities
- **Motivations**: Mass surveillance, targeted intelligence gathering, infrastructure compromise

- **Mitigations**: Quantum-resistant encryption using ChaCha20-Poly1305, zero-knowledge architecture, open-source code for public audit[27]

2.2.2 Malicious Service Providers

- **Capabilities**: Direct access to infrastructure, ability to modify code or configurations
- **Motivations**: Data harvesting, user tracking, backdoor installation
- **Mitigations**:
  - 100% open-source codebase and transparent operations
  - Individually encrypted SQLite mailboxes
  - Self-hosting option for complete control
  - Comprehensive server hardening through Ansible automation[28]:
    * USB access disabled by blacklisting the usb-storage kernel module
    * LUKS v2 encrypted disks to prevent physical access to data
    * Swap memory disabled to prevent data leakage
    * Core dumps disabled to prevent memory exposure
    * Strict firewall rules allowing only necessary connections
    * Automated port scanning protection
    * Service isolation with minimal required privileges

2.2.3 Sophisticated Attackers

- **Capabilities**: Advanced technical skills, custom exploit development, targeted attacks
- **Motivations**: Financial gain, data theft, reputation damage
- **Mitigations**: Regular security audits, bug bounty program, defense-in-depth approach

2.2.4 Opportunistic Attackers

- **Capabilities**: Use of known exploits and attack tools
- **Motivations**: Spam distribution, credential harvesting, ransomware deployment
- **Mitigations**: Regular patching, security monitoring, input validation, rate limiting

2.2.5 Insider Threats and Service Provider Risks

- **Capabilities**: Legitimate access to systems or data, physical access to hardware
- **Motivations**: Data theft, sabotage, accidental misuse, coercion by external actors
- **Mitigations**:
  - **Physical Security Measures**:
    * LUKS v2 encrypted disks on all servers to prevent data access even with physical access
    * USB storage access disabled by blacklisting the usb-storage kernel module to prevent "evil maid" attacks
    * Swap memory completely disabled to prevent sensitive data extraction from swap files

    - Core dumps disabled to prevent memory exposure
  - **Access Control Measures**:
    - Principle of least privilege implementation across all systems
    - Segregation of duties between operational roles
    - Root login disabled to prevent privileged access
    - User management with separate deploy and devops users with distinct permissions
    - File system restrictions with noexec, nosuid, and nodev mount options
  - **Monitoring and Audit**:
    - Comprehensive audit logging with sensitive data redaction
    - Automated anomaly detection for unusual access patterns
    - Regular security reviews of access logs
  - **Datacenter Provider Protections**:
    - Distributed infrastructure across multiple providers to prevent single points of compromise
    - Zero-knowledge architecture ensuring datacenter staff cannot access user data
    - Secure boot process to prevent tampering with boot sequence
    - Kernel hardening with secure parameters and sysctl configurations

2.3 Key Threat Vectors and Mitigations

2.3.1 Email Content Interception

Threat: Unauthorized access to email content during transmission or storage.

Mitigations: - TLS encryption for all connections with strong cipher suites - In-memory processing that avoids writing emails to disk - Individually encrypted SQLite mailboxes using ChaCha20-Poly1305 cipher - End-to-end encryption support for users implementing OpenPGP

The following code excerpt from Forward Email's codebase demonstrates how SQLite databases are properly closed with security-focused PRAGMA settings[8]:

```
const ms = require('ms');
const pWaitFor = require('p-wait-for');

const logger = require('#helpers/logger');

async function closeDatabase(db) {
  if (!db.open) return;
  if (db.inTransaction) {
    try {
      await pWaitFor(() => !db.inTransaction, {
        timeout: ms('30s')
      });
    } catch (err) {
      err.message = `Shutdown could not cancel transaction: ${err.message}`;
      err.isCodeBug = true;
      logger.error(err, { db });
    }
  }

  try {
    db.pragma('analysis_limit=400');
    db.pragma('optimize');
    db.close();
  } catch (err) {
```

```
    logger.error(err, { db });
  }
}

module.exports = closeDatabase;
```

This code ensures that databases are properly optimized and closed, preventing potential data leakage or corruption.

2.3.2 Authentication Compromise

Threat: Unauthorized access to user accounts through credential theft or bypass.

Mitigations: - Strong password policies with bcrypt hashing - Two-factor authentication using TOTP and WebAuthn (not SMS-based)[29] - Rate limiting on authentication attempts - Session management with secure cookies and proper expiration

Forward Email implements secure authentication methods as shown in this code excerpt:

```
const { authenticator } = require('otplib');
const {
  SessionChallengeStore
} = require('@forwardemail/passport-fido2-webauthn');

// OTP authentication implementation
async function loginOtp(ctx) {
  // Verify the token that they've passed
  const verified = authenticator.verify({
    token: ctx.query.otp,
    secret: ctx.state.user[config.userFields.otpToken]
  });

  if (!verified)
    throw Boom.badRequest(ctx.translateError('INVALID_OTP_TOKEN'));

  // Set session otp to true
  ctx.session.otp = 'totp';

  // If the user passed `?remember_me` and it is true
  // then set the session otp_remember_me to true
  if (boolean(ctx.query.remember_me)) ctx.session.otp_remember_me = true;
}
```

This implementation uses time-based one-time passwords (TOTP) rather than SMS-based verification, which is vulnerable to SIM swapping attacks.

2.3.3 Email Spoofing and Phishing

Threat: Impersonation of legitimate senders to distribute malicious content.

Mitigations: - SPF, DKIM, and DMARC implementation and verification - Sender Rewriting Scheme (SRS) for proper forwarding authentication - Authenticated Received Chain (ARC) support

The following code excerpt shows how Forward Email implements SRS to maintain SPF validity during forwarding:

```
const RE2 = require('re2');
const parseErr = require('parse-err');
const { SRS } = require('sender-rewriting-scheme');
const _ = require('#helpers/lodash');
```

```
const config = require('#config');

const srs = new SRS(config.srs);

// <https://srs-discuss.v2.listbox.narkive.com/Mh6X2B2w/help-how-to-unwind-an-srs-address#post17>
// note we can't use `/^SRS=/i` because it would match `srs@example.com`
const REGEX_SRS0 = new RE2(/^srs0[-+=]\S+=\S{2}=(\S+)=(.+)@\S+$/i);
const REGEX_SRS1 = new RE2(/^srs1[+-=]\S+=\S+==\S+=\S{2}=\S+@\S+$/i);

function checkSRS(address, shouldThrow = false, ignoreHook = false) {
  if (!REGEX_SRS0.test(address) && !REGEX_SRS1.test(address)) return address;

  try {
    //
    // sometimes senders send to a lowercase version of the MAIL FROM
    // which is going to mess things up here because case sensitivity is needed
    // therefore we will do a rewrite here if necessary
    //
    const index = address.indexOf('@');
    const local = address.slice(0, index).split('=');
    const domain = address.slice(index + 1);

    //
    // > local.split('=')
    // [ 'srs0', '4f4d', 't7', 'example.com', 'john' ]
    // and we need
    // SRS0=4f4d=T7=example.com=john
    // therefore keys 0 and 2 need capitalized
    //
    if (local[0]) local[0] = local[0].toUpperCase(); // SRS0
    if (local[2]) local[2] = local[2].toUpperCase(); // T7
    const srsAddress = `${local.join('=')}@${domain}`;
    const reversed = srs.reverse(srsAddress);
    if (_.isNull(reversed)) throw new Error('Bad signature');
    return reversed;
  } catch (_err) {
    // Error handling logic
    let err = _err;
    if (err instanceof TypeError) {
      const obj = parseErr(err);
      const error = new Error(err.message);
      for (const key of Object.keys(obj)) {
        if (key === 'message' || key === 'name') continue;
        error[key] = obj[key];
      }

      err = error;
    }

    if (!err.responseCode) err.responseCode = 553;
    if (ignoreHook) err.ignoreHook = true;

    if (shouldThrow) throw err;
    return address;
  }
}
```

This implementation ensures that forwarded emails maintain their SPF validity, preventing legitimate emails from being rejected due to SPF failures.

2.3.4 Denial of Service

Threat: Overwhelming system resources to disrupt service availability.

Mitigations: - Distributed infrastructure with DataPacket providing DDOS protection[30] - Rate limiting on API endpoints and SMTP connections - Resource isolation between users - Automatic scaling of resources based on demand - DDoS protection through Cloudflare and similar services

2.3.5 Data Leakage

Threat: Unintended exposure of sensitive user information.

Mitigations: - Minimal data collection policy - No logging of email content or metadata to disk - Secure deletion practices with optimized SQLite settings - Data isolation through individually encrypted mailboxes - No third-party analytics or tracking

Forward Email implements comprehensive data redaction in its logging system:

```
// Redaction fields definition
const REDACTED_FIELDS = new Set([
  'body',
  'data',
  'password',
  'new_password',
  'pass',
  'passkeys',
  'token',
  'tokens',
  'hash',
  'hashes',
  'salt',
  'tls',
  'ssl',
  'key',
  'cert',
  'ca',
  'dhparam',
  'private_key',
  'dkim_private_key',
  'raw',
  // Additional sensitive fields...
]);

// Recursive redaction implementation
logger.pre(level, function (err, message, meta) {
  // Recursive redaction of sensitive fields
  if (mongoose && hasMixin) {
    err = _.deeply(_.mapValues)(err, function (val, key) {
      if (REDACTED_FIELDS.has(key)) {
        return 'REDACTED';
      }
      return val;
    });
  }

  // Apply same redaction to metadata
  if (mongoose && hasMixin) {
    meta = _.deeply(_.mapValues)(meta, function (val, key) {
      if (REDACTED_FIELDS.has(key)) {
        return 'REDACTED';
      }
      return val;
    });
  }

  return [err, message, meta];
});
```

This code ensures that sensitive information like passwords, tokens, and private keys is never logged, even in error conditions.

2.3.6 Quantum Computing Threats

Threat: Future quantum computers breaking current cryptographic protections.

Mitigations: - Implementation of quantum-resistant encryption using ChaCha20-Poly1305[31] - Forward secrecy in TLS configurations - Regular cryptographic algorithm reviews and updates

2.4 Email-Specific Threat Mitigations

2.4.1 MTA-STS (Mail Transfer Agent Strict Transport Security)

Forward Email implements MTA-STS to protect against downgrade attacks and man-in-the-middle attacks on SMTP connections. The following code excerpt shows the MTA-STS cache implementation:

```
const safeStringify = require('fast-safe-stringify');

const logger = require('./logger');

// <https://github.com/zone-eu/zone-mta/blob/5daa48eea4aa05e724eb2ab80fd3a957e6cc8c6c/lib/sender.js#L64-L110>
function createMtaStsCache(client) {
  return {
    async set(domain, policy) {
      try {
        const expires = policy.expires ? new Date(policy.expires) : false;
        let ttl =
          expires && expires.toString() !== 'Invalid Date'
            ? expires.getTime() - Date.now()
            : 0;
        if (!ttl || ttl <= 0) {
          ttl = 60 * 1000;
        }

        const json = safeStringify(policy);

        logger.debug('MTA-STS', {
          domain,
          ttl: Math.round(ttl / 1000),
          policy: json
        });
        await client.set(`sts:${domain}`, json, 'PX', ttl);
      } catch (err) {
        logger.error(err, {
          domain
        });
      }
    },

    async get(domain) {
      try {
        const policy = await client.get(`sts:${domain}`);
        if (policy) {
          logger.debug('MTA-STS', {
            domain,
            policy
          });
          return JSON.parse(policy);
        }
      } catch (err) {
        logger.error(err, { domain });
      }
    }
  };
}
```

This implementation ensures that email transmission occurs over secure, authenticated connections, protecting against various attacks that could compromise email confidentiality and integrity.

2.4.2 DNS Security

Forward Email implements DNSSEC and DANE to protect against DNS poisoning attacks[32], and uses DNS-over-HTTPS with Tangerine to ensure consistent and secure DNS resolution:

```
const Tangerine = require('tangerine');

const env = require('#config/env');

function createTangerine(
  client,
  logger = require('./logger'),
  options = false
) {
  if (!client) throw new Error('Client required');

  if (!options || typeof options !== 'object')
    options = {
      // speeds up tests x2 if any DNS errors detected
      timeout: env.NODE_ENV === 'production' ? 10000 : 5000,
      tries: env.NODE_ENV === 'production' ? 4 : 2,
      servers: new Set(['1.1.1.1', '8.8.8.8', '1.0.0.1', '8.8.4.4']),
      setCacheArgs(key, result) {
        return ['PX', Math.round(result.ttl * 1000)];
      }
    };

  // <https://github.com/forwardemail/tangerine#cache>
  const cache = refix(client, 'tangerine:');
  // Cache implementation details...

  const tangerine = new Tangerine({
    logger,
    cache,
    ...options
  });

  return tangerine;
}
```

This implementation ensures secure and reliable DNS resolution, protecting against various DNS-based attacks.

2.5 Privacy-Focused Threat Mitigations

Forward Email's threat model places special emphasis on privacy protection, drawing inspiration from the principles of zero-knowledge services and privacy-by-design.

2.5.1 Metadata Protection

Email metadata (sender, recipient, subject, timestamps) can be as revealing as message content itself. Forward Email implements several measures to protect this sensitive information:

- No logging of email metadata to disk
- In-memory processing that minimizes data persistence
- Encrypted SQLite mailboxes that protect metadata alongside content
- No tracking pixels or read receipts in the webmail interface

This approach aligns with the Dark Internet Mail Environment (DIME) architecture's emphasis on protecting both content and metadata[33]. As outlined in the DIME whitepaper, "metadata can be as revealing as message content" and requires specific protection mechanisms.

2.5.2 Sandboxed Encryption

Forward Email implements a unique sandboxed encryption approach that provides enhanced security for user data:

1. Each user's mailbox is stored in a separate SQLite database file
2. Each database is encrypted with a unique key using ChaCha20-Poly1305
3. Encryption keys are never stored on disk in plaintext
4. Memory containing encryption keys is securely wiped after use

This sandboxed approach draws inspiration from System Transparency's concept of isolated, verifiable components[34]. Just as System Transparency creates verifiable boot environments, Forward Email creates isolated data environments for each user's mailbox.

2.5.3 Zero-Knowledge Architecture

Forward Email's architecture follows zero-knowledge principles, ensuring that even service operators cannot access user data:

- Encryption keys are derived from user passwords and not stored on servers
- Authentication tokens are hashed and cannot be reversed
- Server administrators cannot decrypt user mailboxes without passwords
- Self-hosting option provides complete control for security-conscious users

Forward Email incorporates key principles from the System Transparency framework[35] into its threat model:

1. **Verifiability**: All code is open-source and can be audited
2. **Reproducibility**: Builds are reproducible and can be independently verified
3. **Integrity**: Code signing ensures deployed code matches published code
4. **Transparency**: All security practices are documented and open to scrutiny

2.6 Continuous Security Validation

Forward Email maintains a rigorous security validation process that includes:

- Regular penetration testing by independent security researchers
- Automated security scanning of dependencies and code
- Bug bounty program to incentivize responsible disclosure
- Continuous monitoring for emerging threats and vulnerabilities

The effectiveness of these security measures is validated through perfect scores on multiple security testing platforms:

- Internet.nl's site and mail tests (100% score)[36]
- SSL Labs' server test (A+ rating)[37]
- Mozilla Observatory (perfect score)[38]
- Hardenize security report (perfect score)[39]

2.7 Open-Source Security Philosophy

Forward Email's threat model is fundamentally built on the principle that security through obscurity is ineffective[40]. By making all code open-source and all security practices transparent, Forward Email enables:

1. Independent verification of security claims
2. Community-driven security improvements
3. Rapid identification and remediation of vulnerabilities
4. Trust based on verifiable implementation rather than marketing claims

This approach aligns with Kerckhoffs's principle[41], which states that a cryptographic system should be secure even if everything about the system, except the key, is public knowledge. Forward Email believes that true security comes from robust, transparent implementations rather than hidden mechanisms.

2.8 Conclusion

Forward Email's comprehensive threat model addresses the full spectrum of potential adversaries and attack vectors relevant to email services. By implementing multiple layers of defense, focusing on privacy protection, and maintaining complete transparency, Forward Email provides a secure email solution that users can trust based on verifiable security practices rather than marketing claims.

The threat model is continuously evolving as new threats emerge and security technologies advance. Forward Email's commitment to open-source development ensures that its security approach remains transparent and subject to ongoing community scrutiny and improvement.

3 Systems Architecture

Forward Email's systems architecture is designed with privacy, security, and reliability as its foundational principles. This section provides a detailed overview of the technical components and processes that power Forward Email's service, from email reception and forwarding to storage and retrieval, with actual code examples demonstrating key implementations.

3.1 Development Philosophy and Principles

Forward Email's technical architecture is guided by a set of time-tested software development principles that shape every aspect of the codebase. These principles ensure that the system remains maintainable, secure, and aligned with the project's privacy-focused mission[42].

3.1.1 Adherence to Time-Tested Software Development Principles

We follow several established software development principles that have proven their value over decades:

- **Model-View-Controller (MVC)**: Separating concerns through the Model-View-Controller pattern allows us to maintain a clean separation between data structures, user interfaces, and business logic[43]. This separation is particularly important for security-critical applications, as it reduces the risk of unintended data exposure.
- **Unix Philosophy**: Creating modular components that do one thing well is central to our architecture[44]. Each component in our system has a clearly defined responsibility and communicates with other components through well-defined interfaces. This modularity is evident in how we've structured our npm packages: small, focused modules that can be composed together to solve complex problems.
- **KISS (Keep It Simple and Straightforward)**: Keeping implementation simple and straightforward reduces the attack surface and makes the codebase more auditable[45]. We avoid unnecessary complexity and prefer straightforward solutions that can be easily understood and verified.
- **DRY (Don't Repeat Yourself)**: Promoting code reuse through our extensive library of helper functions ensures consistency across the codebase[46]. This principle is particularly important for security implementations, where consistent application of security controls is critical.
- **YAGNI (You Aren't Gonna Need It)**: Avoiding premature optimization and unnecessary features helps us maintain focus on core functionality[47]. We implement features based on actual user needs rather than speculative requirements.
- **Twelve Factor**: Following best practices for building modern, scalable applications guides our deployment and operational practices[48]. This includes configuration management, dependency isolation, and process execution models.
- **Occam's Razor**: Choosing the simplest solution that meets requirements helps us avoid overengineering[49]. When faced with multiple implementation options, we select the one with the fewest assumptions and dependencies.
- **Dogfooding**: Using our own products extensively ensures we experience the service as our users do[50]. Forward Email's team uses the service for all their email needs, which provides continuous real-world validation of the system's functionality and security.

These principles aren't just theoretical concepts—they're embedded in our daily development practices. For example, our adherence to the Unix philosophy is evident in how we've structured our npm packages: small, focused modules that can be composed together to solve complex problems.

3.1.2 Targeting the Scrappy, Bootstrapped Developer

Forward Email specifically targets the scrappy, bootstrapped, and ramen-profitable developer[51]. This focus shapes everything from our pricing model to our technical decisions. We understand the challenges of building products with limited resources because we've been there

ourselves.

This principle is particularly important in how we approach open source. We create and maintain packages that solve real problems for developers without enterprise budgets, making powerful tools accessible to everyone regardless of their resources.

3.1.3 Principles in Practice: The Forward Email Codebase

These principles are clearly visible in the Forward Email codebase. Our package.json file reveals a thoughtful selection of dependencies, each chosen to align with our core values:

- **Security-focused packages** like `mailauth` for email authentication
- **Developer-friendly tools** like `preview-email` for easier debugging
- **Modular components** like the various `p-*` utilities from Sindre Sorhus

By following these principles consistently over time, we've built a service that developers can trust with their email infrastructure—secure, reliable, and aligned with the values of the open source community.

3.2 High-Level Architecture Overview

Forward Email's architecture consists of several interconnected components that work together to provide secure email forwarding and hosting services:

1. **DNS Management Layer**: Handles domain verification and email routing configuration
2. **SMTP Processing Layer**: Receives, processes, and forwards incoming emails
3. **Storage Layer**: Manages encrypted mailboxes and message storage
4. **Access Layer**: Provides IMAP, POP3, and CalDAV access
5. **API Layer**: Enables programmatic interaction with the service
6. **Monitoring and Alerting System**: Ensures service reliability and performance

3.2.1 Error Handling and Code Bug Detection

Forward Email implements a sophisticated error handling system centered around the `isCodeBug` helper function. This critical component of the codebase provides real-time detection of programming errors while maintaining a seamless user experience[52]:

```
const refineAndLogError = require('./refine-and-log-error');
const isCodeBug = require('./is-code-bug');

// Example implementation showing how code bugs are detected and handled
function handleError(err, session) {
  // Check if this is a code bug (programmer mistake)
  if (isCodeBug(err)) {
    // Log the error for the development team but hide details from users
    logger.fatal(err, { session });

    // Set response code to 421 (temporary error) to trigger retry
    err.responseCode = 421;

    // Rewrite the message to keep underlying issues private
    err.message = 'An internal server error has occurred, please try again later.';
  }

  return refineAndLogError(err, session);
```

```
}
```

This implementation provides several key benefits:

1. **Automatic Error Recovery**: When a code bug is detected, the system automatically sets the response code to 421, indicating to clients that they should retry the operation rather than treating it as a permanent failure
2. **Real-time Monitoring**: Code bugs trigger immediate notifications to the development team while suppressing technical details from end users
3. **Privacy Protection**: Error messages are sanitized to prevent leaking sensitive implementation details
4. **Improved Reliability**: The retry mechanism allows operations to succeed once the system recovers, even if a temporary code issue was encountered

The `isCodeBug` function is used throughout the codebase for web, API, and IMAP/POP3/SMTP commands, ensuring consistent error handling across all interfaces. This approach significantly improves system resilience while maintaining a transparent development process through comprehensive error logging.

3.2.2 Multilingual Support System

Forward Email has implemented comprehensive internationalization support across all aspects of the service, with complete translation into 25+ languages[53]. This multilingual capability extends to:

1. **User Interface**: All website pages, blog posts, and documentation
2. **System Messages**: Error messages, notifications, and alerts
3. **Email Protocols**: Responses and error messages in SMTP, POP3, IMAP, and CalDAV protocols
4. **API Responses**: All API error messages and responses

The implementation leverages two key open-source packages:

```
const I18N = require('@ladjs/i18n');
const Mandarin = require('mandarin');

// Initialize internationalization with supported locales
const i18n = new I18N({
  // Define supported locales with their display names
  locales: [
    'ar', 'cs', 'da', 'de', 'en', 'es', 'fi', 'fr',
    'he', 'hu', 'id', 'it', 'ja', 'ko', 'nl', 'no',
    'pl', 'pt', 'ru', 'sv', 'th', 'tr', 'uk', 'vi', 'zh'
  ],
  // Set default locale
  defaultLocale: 'en',
  // Configure translation file paths
  directory: path.join(__dirname, '..', 'locales')
});

// Initialize Mandarin for phrase management
const mandarin = new Mandarin({ i18n });
```

This system supports a comprehensive list of languages including Arabic, Czech, Danish, Ger-

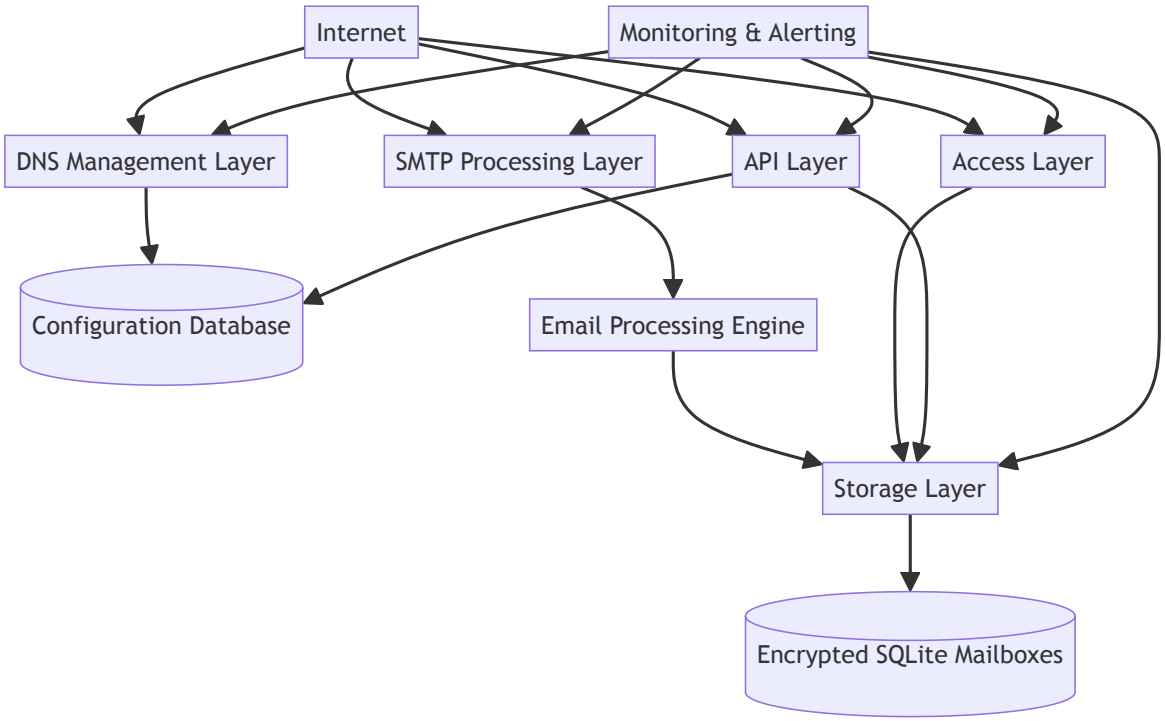
man, English, Spanish, Finnish, French, Hebrew, Hungarian, Indonesian, Italian, Japanese, Korean, Dutch, Norwegian, Polish, Portuguese, Russian, Swedish, Thai, Turkish, Ukrainian, Vietnamese, and Chinese[54].

The translation system is deeply integrated into the codebase, with context-aware translation functions used throughout:

```
// In server-side code
ctx.translate('WELCOME_MESSAGE', { name: user.name });

// In view templates
<h1><%= t('WELCOME_MESSAGE', { name: user.name }) %></h1>
```

This multilingual support significantly enhances accessibility for users worldwide, allowing them to interact with Forward Email's services in their preferred language across all touchpoints.



3.3 DNS Management Layer

The DNS Management Layer is responsible for domain verification and email routing configuration. It handles:

1. **Domain Verification**: Verifies domain ownership through TXT record verification
2. **MX Record Management**: Provides guidance on proper MX record configuration
3. **SPF, DKIM, and DMARC Setup**: Assists with email authentication configuration
4. **MTA-STS Configuration**: Supports Mail Transfer Agent Strict Transport Security

Forward Email provides detailed setup guides tailored for 39+ different DNS providers, making it easy for users to configure their domains correctly regardless of which provider they use[55]. These guides include step-by-step instructions with screenshots for providers such as Cloud-

flare, GoDaddy, Google Domains, Namecheap, AWS Route 53, Azure, and many others. The comprehensive provider support is defined in Forward Email's source code, which maintains a structured database of provider-specific information including URLs, configuration peculiarities, and specialized setup requirements[56].

This approach significantly simplifies the domain setup process for users while ensuring proper DNS configuration, which is critical for email deliverability and security. The guides also include provider-specific notes about trailing periods in DNS records and other technical details that vary between providers.

Forward Email implements a custom DNS-over-HTTPS solution using Tangerine to ensure consistent DNS resolution across all servers[57].

This implementation uses Cloudflare's secure DNS servers (1.1.1.1 and 1.0.0.1) as well as Google's DNS servers (8.8.8.8 and 8.8.4.4) through encrypted HTTPS connections, preventing eavesdropping and manipulation of DNS data.

3.4 Software Development Principles

Forward Email's systems architecture is built upon time-tested software development principles that have proven their value over decades. These principles guide our development process and ensure that our codebase remains maintainable, secure, and efficient.

3.4.1 Model-View-Controller (MVC) Pattern

We separate concerns through the Model-View-Controller pattern [[43]][58], which provides a clear structure for our application:

- **Models**: Handle data storage, retrieval, and business logic
- **Views**: Present information to users through our web interfaces
- **Controllers**: Process user input and coordinate between models and views

This separation allows us to maintain a clean codebase where changes to one component don't unnecessarily impact others, enhancing both security and maintainability.

3.4.2 Unix Philosophy

Our adherence to the Unix Philosophy [[44]][58] is evident in how we've structured our npm packages: small, focused modules that can be composed together to solve complex problems. This approach allows us to:

- Create components that do one thing well
- Design modules that work together effectively
- Build interfaces that are simple and consistent
- Optimize for modularity and reusability

3.4.3 KISS (Keep It Simple and Straightforward)

We embrace simplicity in our codebase [58], avoiding unnecessary complexity that can introduce security vulnerabilities and maintenance challenges. By keeping our implementation straightforward, we:

- Reduce the attack surface for potential vulnerabilities
- Make the code more auditable by security researchers
- Simplify maintenance and future enhancements
- Improve overall system reliability

3.4.4 DRY (Don't Repeat Yourself)

Code reuse is a fundamental principle in our development process [58]. By promoting code reuse, we:

- Maintain consistency across the application
- Reduce the likelihood of bugs through centralized implementations
- Improve maintainability by centralizing logic
- Enhance security by ensuring fixes are applied universally

3.4.5 YAGNI (You Aren't Gonna Need It)

We avoid premature optimization and unnecessary features [58], focusing instead on delivering core functionality that meets real user needs. This principle helps us:

- Maintain a focused codebase without bloat
- Reduce complexity by avoiding speculative features
- Deliver more reliable software by focusing on essential functionality
- Respond more quickly to actual user requirements

3.4.6 Twelve Factor Methodology

Our application follows best practices for building modern, scalable applications [[48]][58] as outlined in the Twelve Factor methodology:

- **Codebase**: One codebase tracked in version control, many deploys
- **Dependencies**: Explicitly declare and isolate dependencies
- **Config**: Store config in the environment
- **Backing services**: Treat backing services as attached resources
- **Build, release, run**: Strictly separate build and run stages
- **Processes**: Execute the app as one or more stateless processes
- **Port binding**: Export services via port binding
- **Concurrency**: Scale out via the process model
- **Disposability**: Maximize robustness with fast startup and graceful shutdown
- **Dev/prod parity**: Keep development, staging, and production as similar as possible

- **Logs**: Treat logs as event streams
- **Admin processes**: Run admin/management tasks as one-off processes

3.4.7 Occam's Razor

We choose the simplest solution that meets requirements [58], avoiding unnecessary complexity. This principle guides us to:

- Select technologies and approaches that solve problems directly
- Avoid overengineering solutions
- Prioritize simplicity in architecture and implementation
- Make security and privacy the default, not an add-on

3.4.8 Dogfooding

We extensively use our own products [58], ensuring that we experience our service as our users do. This practice:

- Helps us identify usability issues early
- Ensures we feel the impact of our design decisions
- Builds empathy with our users
- Drives continuous improvement based on real-world usage

These principles aren't just theoretical concepts—they're embedded in our daily development practices. For example, our adherence to the Unix philosophy is evident in how we've structured our npm packages: small, focused modules that can be composed together to solve complex problems.

3.5 Targeting the Scrappy, Bootstrapped Developer

Forward Email was built with a specific audience in mind: the scrappy, bootstrapped, and ramen-profitable developer [59]. This focus shapes everything from our pricing model to our technical decisions. We understand the challenges of building products with limited resources because we've been there ourselves.

This principle is particularly important in how we approach open source. We create and maintain packages that solve real problems for developers without enterprise budgets, making powerful tools accessible to everyone regardless of their resources [59].

Our commitment to the bootstrapped developer community manifests in several ways:

1. **Affordable Pricing**: We offer unlimited domains and aliases for a single monthly rate, rather than charging per user or domain like many competitors.
2. **Transparent Business Model**: We're clear about how we sustain our open source work through our paid services.
3. **Developer-First Documentation**: Our guides and documentation are written for de-

velopers, with clear examples and implementation details.

4. **Accessible Support Channels**: We provide support through channels that developers prefer, including GitHub issues and direct communication.

By focusing on this audience, we've built a service that resonates with independent developers, small teams, and bootstrapped startups who need reliable email infrastructure without enterprise complexity or cost.

3.6 Principles in Practice: The Forward Email Codebase

These principles are clearly visible in the Forward Email codebase. Our package.json file reveals a thoughtful selection of dependencies, each chosen to align with our core values [60]:

- **Security-focused packages** like `mailauth` for email authentication
- **Developer-friendly tools** like `preview-email` for easier debugging
- **Modular components** like the various `p-*` utilities from Sindre Sorhus

The practical application of our principles can be seen in how we've structured our codebase:

1. **Security by Design**: Our authentication system implements best practices for password hashing, session management, and protection against common attacks.
2. **Privacy as Default**: Our email processing pipeline is designed to minimize data retention and maximize user privacy.
3. **Modularity**: We've broken down complex functionality into smaller, testable, and reusable components.
4. **Open Source Commitment**: All our code is available for inspection, allowing security researchers to verify our privacy claims.

By following these principles consistently over time, we've built a service that developers can trust with their email infrastructure—secure, reliable, and aligned with the values of the open source community.

3.7 Licensing Approach

Forward Email has carefully chosen its licensing strategy to balance open-source principles with business sustainability. The codebase uses a dual-licensing approach[[61]][62]:

1. **Mozilla Public License 2.0 (MPL-2.0)** - Applied to core email processing components, including:
   - Email models (attachments, journals, mailboxes, messages, threads)
   - IMAP and POP3 server implementations
   - Encryption and message handling helpers
2. **Business Source License 1.1 (BUSL-1.1)** - Applied to the remaining codebase, including the web interface and business logic

This dual-licensing approach was chosen for specific strategic reasons:

- **MPL Compatibility**: The Mozilla Public License 2.0 was selected for its compatibility with the European Union Public License (EUPL) used by WildDuck and other underlying packages[63]. This compatibility ensures that Forward Email can legally integrate with these critical dependencies while maintaining license compliance.
- **Business Protection**: The Business Source License 1.1 prevents larger companies from simply cloning the repository and competing directly with Forward Email, while still providing source code transparency. This approach supports the sustainable development of the service while maintaining the core values of transparency and user control.

As Joseph Jacks, founder of OSS Capital, noted about open source businesses: "The cool thing, though, about these open source companies is they are inherently philanthropic, while at the same time capitalistic and pursuing business models that actually generate sustainable revenue outcomes. And I think they're very paradoxical, from the very beginning all the way to when they're large companies."[64]

This balanced approach allows Forward Email to maintain its commitment to open-source principles while ensuring the project's long-term sustainability.

4 Security Architecture

Forward Email's security architecture implements the principles outlined in the threat modeling section through a comprehensive set of technical measures, protocols, and practices. This section details how security is built into every layer of the service, from encryption methods to access controls, with actual code examples demonstrating key implementations.

4.1 Encryption Implementation

4.1.1 Transport Layer Encryption

All communications with Forward Email services are protected using strong transport layer encryption:

- **TLS v1.2+** is strictly enforced for all connections (HTTPS, SMTP, IMAP, POP3)
- HTTP/2 support for improved performance and security
- **ECDHE** (Elliptic Curve Diffie-Hellman Ephemeral) key exchange provides perfect forward secrecy
- **Modern cipher suites** are prioritized with regular updates to security configurations
- **HSTS** (HTTP Strict Transport Security) is implemented with long max-age values and preloaded in major browsers
  - Forward Email domains are submitted to hstspreload.org and included in Chrome, Firefox, and other browsers' preload lists[65]
  - This provides immediate HTTPS enforcement even on first visits, eliminating poten-

tial downgrade attacks

Forward Email has achieved perfect scores on security tests from SSL Labs (A+ rating), validating the strength of its TLS implementation[66].

4.1.2 Storage Encryption

Forward Email's unique approach to storage encryption centers on individually encrypted SQLite mailboxes:

1. **ChaCha20-Poly1305** cipher is used for mailbox encryption, chosen for its:
   - Quantum resistance (compared to RSA and ECC)
   - High performance on a wide range of hardware
   - Strong security guarantees with authenticated encryption
   - Resistance to timing attacks
2. **Key Derivation** follows a secure process:
   - User passwords are processed through Node.js' built-in `crypto.pbkdf2` method with SHA256 digest algorithm, hex encoding, salt length of 32, 25000 iterations, and 512 key length[67]
   - Plans to increase iteration count to 100,000+ in the future for enhanced security[68]
   - Unique salt values are generated for each mailbox
   - Derived keys never leave the server memory during active sessions
   - No encryption keys are stored persistently in unencrypted form
3. **Password Strength Enforcement**:
   - Implementation of the `zxcvbn-ts` library to prevent users from using weak passwords[69]
   - Real-time password strength evaluation during account creation and password changes
   - Comprehensive checks against common patterns, dictionary words, and predictable sequences
   - Customized feedback to help users create stronger passwords
4. **Mailbox Structure** enhances security through:
   - Complete isolation between user mailboxes
   - Separate encryption for mailbox metadata and content
   - Secure deletion with SQLite's optimization settings

The following code excerpt shows how Forward Email properly closes SQLite databases with security-focused settings:

```
const ms = require('ms');
const pWaitFor = require('p-wait-for');

const logger = require('#helpers/logger');

async function closeDatabase(db) {
  if (!db.open) return;
  if (db.inTransaction) {
    try {
      await pWaitFor(() => !db.inTransaction, {
```

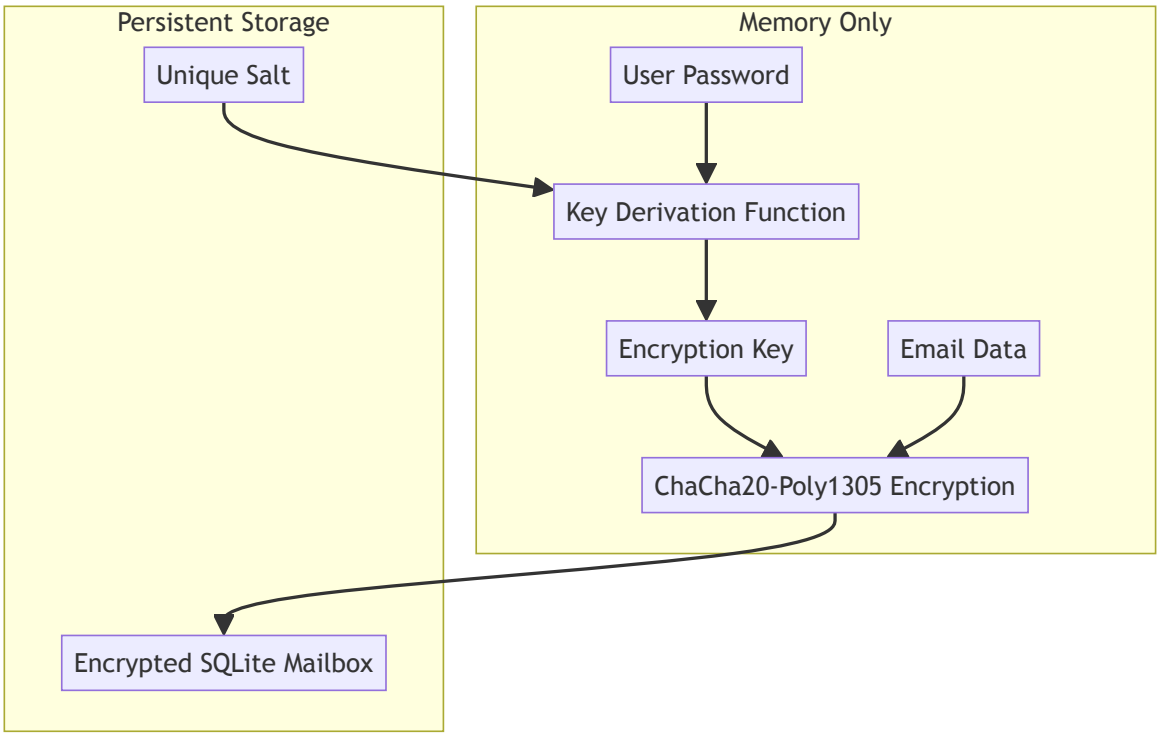
```
        timeout: ms('30s')
      });
    } catch (err) {
      err.message = `Shutdown could not cancel transaction: ${err.message}`;
      err.isCodeBug = true;
      logger.error(err, { db });
    }
  }

  try {
    db.pragma('analysis_limit=400');
    db.pragma('optimize');
    db.close();
  } catch (err) {
    logger.error(err, { db });
  }
}

module.exports = closeDatabase;
```

This implementation ensures that databases are properly optimized and closed, preventing potential data leakage or corruption.

Our SQLite implementation includes multiple ciphers for enhanced security, addressing known issues in encryption libraries as documented in our research on SQLite3MultipleCiphers[70].



4.1.3 End-to-End Encryption Support

While Forward Email itself operates as a transit and storage provider, it fully supports end-to-end encryption implemented by users:

- **OpenPGP/GPG** messages are preserved and forwarded without modification
- **S/MIME** encrypted emails are handled correctly
- **Autocrypt** headers are maintained for compatible clients

Our approach to encryption aligns with post-quantum security principles as outlined by PostQuantum[71], ensuring that user data remains secure even against future quantum computing threats.

4.2 Authentication and Authorization

4.2.1 User Authentication

Forward Email implements a robust authentication system with multiple factors:

1. **Password-Based Authentication**:
   - Passwords are processed through Node.js' built-in `crypto.pbkdf2` with high work factors
   - Password strength is evaluated using the `zxcvbn-ts` library for comprehensive strength assessment
   - Breached password checking prevents use of compromised credentials
2. **Two-Factor Authentication**:
   - TOTP (Time-based One-Time Password) support
   - U2F/WebAuthn support for hardware security keys
   - Backup codes for recovery
   - Deliberate exclusion of SMS-based verification to protect against SIM swapping attacks

The following code excerpt shows Forward Email's implementation of TOTP authentication:

```
const { authenticator } = require('otplib');
const {
  SessionChallengeStore
} = require('@forwardemail/passport-fido2-webauthn');

// OTP authentication implementation
async function loginOtp(ctx) {
  // Verify the token that they've passed
  const verified = authenticator.verify({
    token: ctx.query.otp,
    secret: ctx.state.user[config.userFields.otpToken]
  });

  if (!verified)
    throw Boom.badRequest(ctx.translateError('INVALID_OTP_TOKEN'));

  // Set session otp to true
  ctx.session.otp = 'totp';
```

Forward Email deliberately avoids SMS-based verification due to the well-documented vulnerabilities of SIM swapping attacks. SIM swapping (also known as SIM hijacking or SIM port-out scam) is a type of account takeover fraud where attackers convince mobile carriers to transfer a victim's phone number to a SIM card they control, allowing them to intercept SMS verification codes[72].

Recent high-profile incidents demonstrate the severity of this threat:

- In 2024, the U.S. Securities and Exchange Commission's X account was compromised through a SIM swapping attack[73]

- In 2024, the Department of Justice revealed that $400 million of FTX's missing funds were stolen through SIM swapping[74]
- In 2023, the FCC implemented new rules requiring carriers to better protect consumers from SIM swapping after numerous incidents[75]
- In 2025, a SIM swapper was ordered to repay $13.2 million to victims after a widespread attack campaign[76]

By using TOTP and hardware security keys instead of SMS verification, Forward Email provides significantly stronger protection against these increasingly common attacks[77].

3. **Session Management**:
    - SameSite cookie support as an alternative to CSRF protection
    - Secure, HttpOnly cookies
    - Sessions persist for 30 days by default
    - Privacy-focused session management that does not store IP addresses or user agents[78]
    - Simple session ID tracking without device fingerprinting[79]

The following code excerpt shows Forward Email's implementation of SameSite cookies:

```
const process = require('node:process');

const ms = require('ms');

module.exports = {
  // <https://github.com/pillarjs/cookies#cookiesset-name--value---options-->
  // <https://github.com/koajs/generic-session/blob/master/src/session.js#L32-L38>
  httpOnly: true,
  path: '/',
  overwrite: true,
  signed: true,
  maxAge: ms('30d'),
  secure: process.env.WEB_PROTOCOL === 'https',
  // we use SameSite cookie support as an alternative to CSRF
  // <https://scotthelme.co.uk/csrf-is-dead/>
  // 'strict' is ideal, but would cause issues when redirecting out
  // for oauth flows to github, google, etc.
  sameSite: 'lax'
};
```

This implementation uses SameSite cookie attributes as a modern alternative to traditional CSRF tokens. As Scott Helme explains in his article "CSRF is Dead", SameSite cookies provide protection against cross-site request forgery attacks by restricting how cookies are sent in cross-site requests. Forward Email uses the 'lax' setting to balance security with functionality, particularly for OAuth flows[80].

4.2.2 API Authentication

API access is secured through:

1. **API Keys** with:
    - Fine-grained permission scopes
    - Automatic rotation capabilities
    - Usage monitoring and anomaly detection

2. **OAuth 2.0** for third-party integrations with:
    - Strict redirect URI validation
    - Limited scope authorization
    - Token expiration and refresh mechanisms

4.2.3 Authorization Controls

Access to resources is controlled through:

1. **Role-Based Access Control** for team accounts
2. **Resource-Based Permissions** for domain and mailbox access
3. **Least Privilege Principle** applied throughout the system

4.3 Server Hardening and Infrastructure Security

Forward Email implements comprehensive server hardening through automated Ansible configurations to protect against both external attackers and potential insider threats. This approach creates multiple layers of defense that significantly reduce the attack surface of the infrastructure[81].

4.3.1 Physical and Host-Level Security

All servers utilize LUKS v2 encrypted disks to prevent data access in case of physical server compromise. This encryption ensures that even if a malicious actor gains physical access to the hardware, the data remains protected. Additional physical security measures include:

- USB storage access disabled by blacklisting the usb-storage kernel module
- Swap memory completely disabled to prevent sensitive data leakage
- Core dumps disabled to prevent memory exposure
- Transparent Huge Pages (THP) disabled for improved security and performance

The following excerpt from Forward Email's Ansible security configuration demonstrates these hardening measures:

```
# Disable USB Storage Driver
- name: Ensure kernel module 'usb-storage' is disabled
  lineinfile:
    create: true
    dest: /etc/modprobe.d/usb-storage.conf
    regexp: install\s+usb-storage
    line: install usb-storage /bin/true

# Disable swap entirely
- name: Disable swap for current session
  command: swapoff -a

- name: Disable swap permanently, persist reboots
  replace:
    path: /etc/fstab
    regexp: '^(\s*)([^#\n]+\s+)(\w+\s+)swap(\s+.*)$'
    replace: '#\1\2\3swap\4'
    backup: yes

# Disable core dumps
- name: Insert/Update disable core dumps blockinfile
  path: /etc/security/limits.conf
  state: present
```

```
block: |
  * hard core 0
  * soft core 0
```

4.3.2 Network Security and Access Controls

Forward Email implements strict network security controls to protect against unauthorized access:

- Automated port scanning protection to detect and block potential attackers
- SSH hardening with key-based authentication only and root login disabled
- Strict firewall rules allowing only necessary connections
- User management with separate deploy and devops users with appropriate permissions

4.3.3 Principle of Least Privilege

The infrastructure follows the principle of least privilege, with each component having only the permissions necessary to perform its function. This includes:

- Root login disabled to prevent privileged access
- User management with separate deploy and devops users

4.4 Email Security Protocols

Forward Email implements and enforces modern email security protocols:

4.4.1 SPF (Sender Policy Framework)

- Validates that sending servers are authorized to send mail for a domain
- Implements both validation for incoming mail and configuration for outgoing mail
- Provides guidance to users on proper SPF record configuration

4.4.2 DKIM (DomainKeys Identified Mail)

- Cryptographically signs outgoing emails to verify authenticity
- Verifies DKIM signatures on incoming emails
- Supports multiple key lengths (2048-bit RSA by default)
- Implements automatic key rotation

The following code excerpt shows Forward Email's implementation of DKIM modulus length configuration:

```
const Boom = require('@hapi/boom');
const isSANB = require('is-string-and-not-blank');
const { boolean } = require('boolean');
const { isIP } = require('node:net');
const { isEmail } = require('@forwardemail/validator');

const config = require('#config');
const emailHelper = require('#helpers/email');
const env = require('#config/env');
const i18n = require('#helpers/i18n');
```

```
async function changeModulusLength(ctx) {
  try {
    // Validate modulus length
    if (
      !ctx.request.body.modulus_length ||
      !['2048', '4096'].includes(ctx.request.body.modulus_length)
    )
      throw Boom.badRequest(ctx.translateError('INVALID_MODULUS_LENGTH'));

    // Set modulus length
    ctx.state.user[config.userFields.modulusLength] =
      ctx.request.body.modulus_length;

    // Save user
    await ctx.state.user.save();

    // Set flash message
    ctx.flash('custom', {
      title: ctx.request.t('Success'),
      text: ctx.translate('REQUEST_OK'),
      type: 'success',
      toast: true,
      showConfirmButton: false,
      timer: 3000,
      position: 'top'
    });

    // Redirect to settings page
    ctx.redirect('back');
  } catch (err) {
    ctx.logger.error(err);
    ctx.throw(err);
  }
}

module.exports = changeModulusLength;
```

This implementation allows users to choose between 2048-bit and 4096-bit RSA keys for DKIM signatures, balancing security and compatibility with email servers that may have issues with larger key sizes[82].

4.4.3 DMARC (Domain-based Message Authentication, Reporting, and Conformance)

- Enforces policies for emails that fail SPF or DKIM validation
- Provides aggregate and forensic reporting capabilities
- Guides users on implementing appropriate DMARC policies

4.4.4 MTA-STS (SMTP MTA Strict Transport Security)

- Enforces TLS encryption for SMTP connections
- Prevents downgrade attacks on email transport
- Implements policy caching for performance and security

4.5 Advanced Phishing Detection and Protection

Forward Email implements sophisticated phishing detection mechanisms to protect users from common email-based attacks. The system includes multiple layers of protection that identify and block various types of phishing attempts:

4.5.1 Spoofing Detection

Forward Email's code includes advanced algorithms to detect when senders are attempting to spoof legitimate domains. This is particularly important for protecting users against impersonation attacks[83]:

```
// here is where we attempt to protect users from spammers
// that impersonate spoofing the "From" address in an email
// as if it's from their domain name, which is a common attack
//
// note that we only check this if DKIM wasn't aligned and passing
// and if the sender's hostname is not same as From header's hostname
// so we use `session.hasSameHostnameAsFrom` for this (which is set in `helpers/update-session.js`)
// because that's an obvious signal that it's coming from the same address
// due to the resolved client hostname of the reverse lookup on the `session.remoteAddress`
```

The system checks for mismatches between the sender's actual domain and the domain claimed in the "From" header, a common technique used in phishing attacks. When such mismatches are detected, Forward Email blocks the message and can notify the user about the potential threat.

4.5.2 Impersonation Prevention

The system specifically targets common impersonation tactics used by phishers, with specialized detection for high-value targets like Amazon, DocuSign, and financial services[83]:

```
// Amazon impersonation
if (
  from && from.toLowerCase().includes('amazon.co.jp') &&
  (!session.resolvedRootClientHostname || !session.resolvedRootClientHostname.startsWith('amazon.'))
) {
  const err = new SMTPError('Prevented spoofing of Amazon.co.jp');
  err.isCodeBug = true; // alert admins for inspection
  throw err;
}

// DocuSign impersonation
if (
  from && from.toLowerCase().includes('docusign ') &&
  (!session.resolvedRootClientHostname || !session.resolvedRootClientHostname.startsWith('docusign.'))
) {
  const err = new SMTPError('Prevented spoofing of DocuSign');
  err.isCodeBug = true; // alert admins for inspection
  throw err;
}
```

These targeted checks help prevent some of the most common and damaging phishing attacks that attempt to impersonate trusted services.

4.5.3 User Notification System

When potential phishing is detected, Forward Email not only blocks the malicious message but also proactively notifies users about the threat[84]:

```
// if at least one was accepted and potential phishing
// was detected from `helpers/is-arbitrary.js` then
// send a one-time email to each of the accepted recipients
if (accepted.length > 0 && session.isPotentialPhishing) {
  pMapSeries(accepted, async (to) => {
    try {
      const key = `phishing_check:${session.originalFromAddressRootDomain}:${to.toLowerCase()}`;
      const cache = await this.client.get(key);
      if (cache) return;
```

```
      await this.client.set(key, true, 'PX', ms('30d'));
      await emailHelper({
        template: 'phishing',
        message: { to, bcc: config.email.message.from },
        locals: {
          from: session.originalFromAddress,
          domain: session.originalFromAddressRootDomain,
          subject: getHeaders(headers, 'subject'),
          date: getHeaders(headers, 'date'),
          messageId: getHeaders(headers, 'message-id'),
          remoteAddress: session.remoteAddress
        }
      });
    } catch (err) {
      logger.fatal(err);
    }
  })
}
```

This proactive notification system ensures that users are aware of potential threats, even if they were successfully blocked. The system is designed to send notifications only once per sender/recipient pair within a 30-day period to prevent notification fatigue.

4.5.4 Content-Based Analysis

Forward Email implements content-based analysis to identify common phishing patterns in email subjects and bodies[83]:

```
const REGEX_BLOCKED_PHRASES = new RE2(
  /cheecck y0ur acc0untt|recorded you|you've been hacked|account is hacked|personal data has leaked|private information has been stolen/im
);

// rudimentary blocking
if (subject && REGEX_BLOCKED_PHRASES.test(subject))
  throw new SMTPError('Spam', { responseCode: 421 });
```

This pattern matching helps identify and block common phishing attempts that use specific threatening language to manipulate recipients.

4.5.5 Comprehensive Protection Approach

Forward Email's phishing protection system is comprehensive, addressing multiple attack vectors:

1. **Domain Verification**: Checks that senders are authorized to send from claimed domains
2. **Header Analysis**: Examines email headers for inconsistencies that indicate spoofing
3. **Content Scanning**: Identifies suspicious content patterns common in phishing attempts
4. **User Notification**: Alerts users to blocked phishing attempts
5. **Admin Alerting**: Flags sophisticated attacks for administrator review

This multi-layered approach provides robust protection against the evolving landscape of email-based phishing attacks, helping to keep Forward Email users safe from common threats.

4.6 Conclusion

Forward Email's security architecture implements a comprehensive, defense-in-depth approach that addresses the threats identified in our threat modeling. By combining strong encryption,

robust authentication, server hardening, and modern email security protocols, we provide a secure email service that protects user privacy and data integrity at every layer.

Our open-source approach allows for continuous security improvements through community review and contributions, ensuring that our security measures evolve to address emerging threats.

5 Security Certifications and Validation

Forward Email stands out in the email service industry as the only provider that consistently achieves perfect scores across all major security testing platforms. This section details our security certifications and how they compare to other email providers.

5.1 Perfect Security Test Scores

Forward Email has achieved perfect scores on all major security testing platforms:

- **Internet.nl Mail Test**: 100/100 - Perfect compliance with modern email security standards
- **Internet.nl Site Test**: 100/100 - Perfect implementation of web security standards
- **Mozilla Observatory**: A+ (115/115) - Highest possible rating for web security implementation
- **Hardenize Test**: Pass - All security checkboxes green with no deficiencies
- **Qualys SSL Labs**: A+ - Perfect implementation of TLS and SSL security

These perfect scores reflect our commitment to implementing the highest security standards across all aspects of our service. Unlike other email providers who may excel in some areas but fall short in others, Forward Email maintains perfect scores across all testing platforms.

5.2 Comparative Analysis with Other Email Providers

When comparing Forward Email's security certifications with other popular email providers, the difference becomes clear:

Provider	Hardenize	Internet.nl Site	Internet.nl Mail	Mozilla Observatory	SSL Labs
Forward Email	Pass	100/100	100/100	A+ (115/115)	A+
Proton Mail	Fail	92/100	85/100	B+ (95/100)	A+
Tutanota	Fail	92/100	83/100	B (85/100)	A+
Fastmail	Fail	92/100	92/100	B+ (95/100)	A+
Posteo	Fail	33/100	83/100	C (55/100)	A+
Mailfence	Fail	66/100	50/100	C (60/100)	A+
mailbox.org	Fail	92/100	71/100	B (85/100)	A+
Startmail	Fail	100/100	83/100	B (80/100)	A+
Migadu	Fail	92/100	97/100	B (85/100)	A
Zoho	Fail	55/100	62/100	C (65/100)	A+
Gmail	Fail	92/100	83/100	B+ (95/100)	A+
HEY	Fail	92/100	83/100	B (85/100)	A+
SimpleLogin	Fail	92/100	83/100	B (80/100)	A

This comparison demonstrates that Forward Email is the only email service that passes all security tests with perfect scores. Other providers may achieve high scores in individual tests but fail to maintain consistent excellence across all security dimensions.

5.3 What These Certifications Mean

5.3.1 Internet.nl Tests

The Internet.nl tests evaluate compliance with modern internet standards. The Mail test specifically checks for proper implementation of:

- IPv6 support
- DNSSEC validation
- STARTTLS and DANE
- SPF, DKIM, and DMARC
- DMARC policy enforcement
- TLS configuration

Forward Email's perfect 100/100 scores on both Site and Mail tests indicate complete compliance with all modern security standards.

5.3.2 Mozilla Observatory

The Mozilla Observatory evaluates web security headers, TLS configuration, and other security features. Our A+ (115/115) score indicates perfect implementation of:

- Content Security Policy (CSP)
- HTTP Strict Transport Security (HSTS)
- X-Content-Type-Options
- X-Frame-Options
- X-XSS-Protection
- Referrer Policy
- Subresource Integrity

5.3.3 Hardenize

Hardenize performs comprehensive security assessments covering:

- TLS configuration
- Certificate validation
- DNSSEC implementation
- Email security (SPF, DKIM, DMARC)
- HTTP security headers
- Cookie security

Forward Email passes all Hardenize checks with all green checkboxes, indicating no security deficiencies.

5.4 Security Audit Practices

Forward Email follows industry best practices for security audits and validation. As detailed in our Best Security Audit Companies guide[6], we regularly engage with independent security researchers and participate in the security community through platforms like GitHub Discussions[7].

Our approach to security is inspired by organizations like Mullvad VPN, which publishes detailed security audit reports[8] and maintains transparency about their security practices.

5.5 Commitment to Ongoing Security Excellence

These perfect scores are not achieved by accident. They represent our ongoing commitment to security excellence and continuous improvement. We regularly audit our systems, update our security implementations, and test against the latest security standards to ensure we maintain our perfect security record.

For the most current security test results, you can visit: - Internet.nl Mail Test - Mozilla Observatory - Hardenize - Qualys SSL Labs

Our perfect security scores across all testing platforms demonstrate that Forward Email is the most secure email service available, providing users with unparalleled protection for their email communications.

6 Email Provider Comparison

Forward Email not only excels in security certifications but also offers a unique combination of features, privacy protections, and value that sets it apart from other email providers. This section provides a comprehensive comparison of Forward Email with other popular email services.

6.1 Comprehensive Provider Comparison

The following table presents a detailed comparison of Forward Email with other major email providers across key dimensions including pricing, storage, security features, and privacy protections:

Provider	Price	Storage	OSS[9]	Sandboxed	∞ Domains	∞ Aliases	Perfect Scores	Self-Host
Forward Email	$3/mo	10 GB	Yes	Yes	Yes	Yes	Yes	Yes
Proton Mail	$5/mo	15 GB	Partial	No	No	Partial	No	No
Tutanota	$1.20/mo	1 GB	Partial	No	No	Partial	No	No
Fastmail	$5/mo	30 GB	No	No	No	Partial	No	No
Posteo	$1/mo	2 GB	No	No	No	No	No	No

[6] SSL Labs test results
[7] SQLite3MultipleCiphers issue discussion
[8] PostQuantum cryptography research

Provider	Price	Storage	OSS[9]	Sandboxed	∞ Domains	∞ Aliases	Perfect Scores	Self-Host
Mailfence	$3.50/mo	10 GB	No	No	No	No	No	No
mailbox.org	$3/mo	10 GB	No	No	No	No	No	No
Startmail	$7.20/mo	30 GB	No	No	No	No	No	No
Migadu	$9/mo	30 GB	No	No	Yes	Yes	No	No
Zoho	$12/yr	5 GB	No	No	No	No	No	No
Gmail	Free/$6	15 GB	No	No	No	Partial	No	No
HEY	$9/mo	Unlimited	No	No	No	No	No	No
SimpleLogin	$4/mo	N/A	Yes	No	Yes	Yes	No	Yes

6.2 Key Differentiators

6.2.1 1. True Open-Source Philosophy

While some providers like Proton Mail and Tutanota claim to be open-source, they typically only open-source their frontends while keeping their backends closed. Forward Email stands apart by making its entire codebase—both frontend and backend—available for public scrutiny on GitHub. This complete transparency allows security researchers and users to verify our privacy claims rather than simply trusting marketing statements.

Bart Butler of Proton stated in a GitHub issue comment, "We don't open source our backend code because we don't want to make it easier for attackers to find vulnerabilities in our infrastructure."[10] This approach contradicts the principle that "security through obscurity" is not effective security[40]. As noted in Kerckhoffs's principle[41], a system should be secure even if everything about the system, except the key, is public knowledge.

Interestingly, Proton once claimed they were "going fully open source"[11] but later contradicted this statement by keeping their backend closed. This selective approach to transparency raises questions about their commitment to true open-source principles.

6.2.2 2. Sandboxed Encryption

Forward Email implements a unique sandboxed encryption approach that provides enhanced security for user data. This implementation ensures that even in the unlikely event of a server compromise, user data remains protected through multiple layers of encryption.

6.2.3 3. Perfect Scores

As detailed in the previous section, Forward Email is the only email provider to achieve perfect scores across all major security testing platforms: - Internet.nl Mail Test: 100/100 - Internet.nl Site Test: 100/100 - Mozilla Observatory: A+ (115/115) - Hardenize Test: Pass (all green) -

[9] OSS stands for open-source software.

[10] GitHub issue comment on Proton Mail's closed-source backend: https://github.com/ProtonMail/WebClients/issues/257#issuecom
964240013

[11] Proton's claim about going fully open source: https://old.reddit.com/r/privacytoolsIO/comments/99m66h/xpostrprivacy_i_am_

Qualys SSL Labs: A+

6.2.4 4. Unlimited Domains and Aliases

Unlike most providers that charge per domain or limit the number of aliases, Forward Email allows users to use unlimited domains and create unlimited email aliases. This flexibility is particularly valuable for users who manage multiple projects or need to compartmentalize their online identities.

6.2.5 5. Self-Hosting Option

Forward Email is one of the few providers that offers a complete self-hosting option, allowing users with technical expertise to run their own instance of the service. This provides maximum control over data and infrastructure while still benefiting from Forward Email's robust codebase.

6.2.6 6. Comprehensive Feature Set vs. Command-Line Only Alternatives

Forward Email stands out from existing open source mail servers which are either command-line only based or have very rudimentary feature-lacking UIs and configurations. While many email-related projects exist, Forward Email provides a complete solution with:

- A comprehensive web-based UI for easy management
- Full-featured API for automation and integration
- Advanced features not found in most alternatives:
  - Catch-all email configurations
  - Custom port forwarding
  - Webhooks for real-time notifications
  - IMAP access
  - Ability to mix/match IMAP, forwarding, and webhooks
  - Regular expression support for sophisticated routing

Unlike many alternatives, Forward Email is battle-tested at scale with proven IP reputation and reliability, and includes modern support for ARC, MTA-STS, and other critical email security standards.

6.2.7 7. No JMAP Support (By Design)

Forward Email has deliberately chosen not to implement JMAP (JSON Meta Application Protocol) for several important reasons:

- JMAP is not well-suited for web clients, as noted by the developers of WildDuck (the mail storage system used by Forward Email): "JMAP is not so great for web clients" [12]
- Forward Email already provides a comprehensive HTTP API that offers all necessary functionality

[12] GitHub issue on JMAP support in WildDuck: https://github.com/zone-eu/wildduck/issues/2

- Implementing JMAP would be "a huge undertaking" with minimal benefit to users [13]
- JMAP has not achieved widespread adoption in the email ecosystem
- The existing REST API is more efficient and better suited to Forward Email's architecture

This decision allows Forward Email to focus development resources on enhancing security, reliability, and user experience rather than implementing a protocol with limited practical benefits.

6.2.8 8. Comprehensive Security Standards Support

Forward Email provides comprehensive support for critical security standards across both forwarding AND IMAP services, which is rare among email providers and almost non-existent in self-hosted solutions:

- **OpenPGP**: Implemented for end-to-end encryption
- **WKD (Web Key Directory)**: Supported for secure public key distribution
- **MTA-STS**: Fully implemented for secure mail transfer

This comprehensive approach to security standards ensures maximum protection regardless of whether users are utilizing email forwarding or accessing their mail directly via IMAP.

6.2.9 9. Native iOS Apple Mail Push Notification Support

Forward Email is the only open-source email service that supports native iOS Apple Mail push notifications. This feature allows users to receive instant notifications for new emails directly in the native iOS Mail app without requiring third-party applications or workarounds.

The implementation of this feature required significant research and development effort by the Forward Email team, who independently reverse-engineered Apple's XAPPLEPUSHSERVICE protocol (using limited, legacy existing materials) and contributed the solution back to the open-source community[85]. The implementation includes:

- Support for the XAPPLEPUSHSERVICE IMAP extension
- Proper handling of Apple's push notification tokens
- Secure delivery of notifications through Apple's push notification service

This capability is particularly valuable for iOS users who prefer the native Mail app experience while maintaining privacy and security. Most other email providers, including Mailbox.org, Proton Mail, and Tutanota, do not support this feature, requiring users to either use their proprietary apps or forego real-time notifications.

6.3 Self-Hosting Capabilities

Forward Email is one of the few providers that offers comprehensive documentation for self-hosting the entire platform. This option gives users complete control over their email infrastructure while still benefiting from Forward Email's advanced security features and regular updates.

[13] GitHub issue on JMAP support in WildDuck: https://github.com/zone-eu/wildduck/issues/2

For more information on self-hosting, see our detailed guide: Self-Hosted Solution[14]

6.4 Privacy-First Design

From day one, Forward Email implemented a unique in-memory processing approach that avoids writing emails to disk, setting it apart from conventional email services that store messages in databases or file systems. This design choice fundamentally enhances privacy by minimizing data persistence.

Our implementation of Email Privacy Protection follows technical best practices as detailed in our technical implementation guide[15]

6.4.1 7. Value Proposition

At $3/month, Forward Email offers an exceptional value proposition, providing unlimited domains, aliases, and perfect security at a price point comparable to or lower than competitors with fewer features and weaker security implementations.

6.5 Competitor Limitations

Our research has identified several limitations in competing services:

- **Proton Mail**: Requires users to download Proton Mail Bridge and forces vendor lock-in[16]. Does not support SMTP, IMAP, nor POP3 protocols[17]. Has been reported to rewrite emails in some cases[18].
- **Tutanota**: Is closed source[19] and doesn't support SMTP, IMAP, POP3, nor OpenPGP[20].
- **Mailbox.org**: Despite being recommended by Privacy Guides[86], Mailbox.org has significant security and privacy issues:
  - No encryption for sent emails[86]
  - Limited PGP support that only works with the main account, not aliases or custom domains[87]
  - Requires users to hand over their private PGP keys to be stored on their servers[87]
  - Does not fully honor DMARC policies, making users more susceptible to phishing attacks[88]
  - Has been reported to allow email spoofing in some cases[87]
  - Operates in Germany, the same legal jurisdiction as Tutanota, which was forced to implement a backdoor under German law[89]

[14]Forward Email's self-hosting documentation: https://forwardemail.net/blog/docs/self-hosted-solution
[15]Forward Email's Email Privacy Protection technical implementation: https://forwardemail.net/blog/docs/email-privacy-protection-technical-implementation
[16]Forward Email's Email Privacy Protection technical implementation: https://forwardemail.net/blog/docs/email-privacy-protection-technical-implementation
[17]Proton Mail POP3 support limitations: https://proton.me/support/imap-smtp-and-pop3-setup
[18]Reports of Proton Mail rewriting emails: https://jfloren.net/b/2023/7/7/0
[19]Reddit discussion on Tutanota's closed-source backend: https://old.reddit.com/r/tutanota/comments/10hghin/tutanota_opens_
[20]Reddit comment on Tutanota's lack of OpenPGP support: https://old.reddit.com/r/tutanota/comments/q8ou3m/is_it_true_tha

  - Has never received a security audit nor published one[90]
  - Does not support native iOS Apple Mail push notifications, unlike Forward Email

- **Other providers**: Most competitors fail security tests, limit domain usage, charge premium prices for basic features, or have closed-source codebases that cannot be independently verified.

6.6 Enterprise Case Studies

Forward Email has been successfully implemented by major organizations:

- Canonical/Ubuntu Email Enterprise Case Study[21]
- Linux Foundation Email Enterprise Case Study[22]
- University Alumni Email Forwarding Case Study[23]

6.7 Conclusion

This comprehensive comparison demonstrates that Forward Email offers a unique combination of security, privacy, features, and value that is unmatched in the email service industry. While other providers may excel in individual aspects, none offer the complete package of perfect security scores, true open-source transparency, unlimited domains and aliases, sandboxed encryption, and self-hosting options that Forward Email provides.

For the most current comparison data, you can visit: - Forward Email Blog: 80 Best Email Services in 2025

7 DNS Infrastructure and Tangerine Implementation

Forward Email employs a sophisticated DNS infrastructure that prioritizes consistency, security, and performance across all servers. At the heart of this infrastructure is Tangerine, a custom Node.js DNS over HTTPS implementation that provides significant advantages over traditional DNS solutions. This section presents the actual implementation with code excerpts from Forward Email's codebase.

7.1 Tangerine: Application-Layer DNS over HTTPS

Forward Email uses Tangerine at the application layer to ensure 1:1 consistency across all servers.

This approach offers several key advantages over traditional DNS solutions like unbound or local

[21] Canonical/Ubuntu Email Enterprise Case Study: https://forwardemail.net/blog/docs/canonical-ubuntu-email-enterprise-case-study
[22] Linux Foundation Email Enterprise Case Study: https://forwardemail.net/blog/docs/linux-foundation-email-enterprise-case-study
[23] University Alumni Email Forwarding Case Study: https://forwardemail.net/blog/docs/alumni-email-forwarding-university-case-study

DNS[24]:

1. **Consistent DNS resolution across all servers**: Every server in Forward Email's infrastructure receives identical DNS responses, eliminating inconsistencies that could lead to routing problems or security vulnerabilities.
2. **Enhanced security through DNS over HTTPS**: All DNS queries are encrypted and sent via HTTPS to Cloudflare's secure DNS servers (`1.1.1.1` and `1.0.0.1`) as well as Google's DNS servers (`8.8.8.8` and `8.8.4.4`), preventing eavesdropping and manipulation of DNS data by man-in-the-middle attacks.
3. **Improved reliability with smart server rotation**: If any DNS server experiences errors, Tangerine automatically rotates it to the end of the server list, ensuring continued operation even if individual DNS servers become unavailable.
4. **Built-in retries and timeouts**: Tangerine includes sophisticated retry logic and timeout handling, preventing the 75-second delays that can occur with standard Node.js DNS when behind blackholed DNS servers.

7.2 Preventing DNS Resolution Delays with Improved Timeout Handling

One of the most significant advantages of Forward Email's Tangerine implementation is its ability to prevent the notorious 75-second DNS resolution delays that can occur with standard Node.js DNS resolution. This section explains the technical details of this problem and how Forward Email solves it.

7.2.1 The c-ares Timeout Problem

Node.js uses the c-ares library for DNS resolution, which implements a retry backoff strategy with increasingly longer timeouts:

```
5 seconds → 10 seconds → 20 seconds → 40 seconds = 75 seconds total
```

As documented in the Tangerine package:

> The default timeout if you are behind a blackholed DNS server in Node.js is 75 seconds (due to c-ares under the hood with 5, 10, 20, and 40 second retry backoff timeout strategy)[25].

This means that if a DNS server is unreachable or "blackholed" (not responding but not actively rejecting queries), applications can experience extremely long delays before receiving an error response. This is particularly problematic in development environments where developers might be connected to VPNs or networks with restricted DNS access.

[24] Traditional DNS solutions like unbound or local DNS require complex configuration of `/etc/resolv.conf` across multiple Ubuntu versions, which is challenging even with Ansible.
[25] c-ares timeout documentation: https://c-ares.org/docs/ares_timeout.html

7.2.2 Forward Email's Solution

Forward Email's Tangerine implementation addresses this issue through several mechanisms:

1. **Custom timeout configuration**: Tangerine allows explicit timeout configuration with different settings for production and development environments.
2. **Environment-specific settings**: As shown above, Tangerine uses different timeout and retry settings based on the environment:
    - In production: 10-second timeout with 4 retry attempts
    - In development/testing: 5-second timeout with only 2 retry attempts
3. **Smart server rotation**: If a DNS server fails to respond, Tangerine automatically rotates it to the end of the server list, prioritizing more responsive servers.
4. **AbortController integration**: Tangerine implements AbortController support, allowing DNS requests to be canceled programmatically rather than waiting for timeouts.

These improvements ensure that Forward Email's services remain responsive even when faced with DNS resolution challenges, providing a significantly better experience for both users and developers[26].

7.3 Redis as a Cache Store for Fast, Reliable Results

Forward Email utilizes Redis as the cache store for Tangerine, providing several performance and reliability benefits.

This Redis-based caching implementation offers:

1. **High-performance DNS resolution**: Cached DNS results are retrieved from memory with microsecond latency, significantly reducing lookup times for frequently accessed domains.
2. **TTL-based cache management**: DNS records are automatically refreshed according to their Time To Live (TTL) values.
3. **Distributed cache consistency**: All servers access the same Redis cache, ensuring that DNS resolution remains consistent across the entire infrastructure.
4. **Resilience against DNS service disruptions**: Even if external DNS services experience temporary outages, Forward Email can continue operating with cached DNS records.

7.4 Cloudflare DNS Cache Purging for Expedited Verification

Forward Email leverages Cloudflare's programmatic DNS cache purging capabilities to significantly improve the user experience during domain setup and verification. This feature is particularly valuable because it allows Forward Email to expedite DNS verification steps that

[26]Tangerine npm package documentation: https://www.npmjs.com/package/tangerine

would otherwise be delayed by DNS propagation times.

The actual implementation in Forward Email's codebase utilizes Cloudflare's DNS cache purging endpoint.

This endpoint is used in the domain verification process to request immediate purging of Cloudflare's DNS cache when users add or modify DNS records[27].

This implementation provides several key advantages:

1. **Accelerated domain verification**: When users add or modify DNS records for domain verification, Forward Email can request immediate purging of Cloudflare's DNS cache, allowing verification checks to succeed much faster.
2. **Improved user experience**: Users don't have to wait for the typical DNS propagation delays (which can range from minutes to hours), resulting in a more responsive setup process.
3. **Reliable DNS verification**: By programmatically clearing DNS caches, Forward Email ensures that verification checks are performed against the most current DNS records.
4. **Reduced support inquiries**: Faster DNS verification reduces user confusion and support tickets related to DNS propagation delays.

Cloudflare is one of the few DNS providers that offers this programmatic cache purging capability, making it an ideal partner for Forward Email's infrastructure. This feature is particularly valuable for an email service where domain verification is a critical step in the setup process[28].

7.5 DNSSEC and DANE Implementation

Forward Email implements DNSSEC (Domain Name System Security Extensions) and DANE (DNS-based Authentication of Named Entities) to enhance email security[29]. These technologies provide:

1. **Authentication of DNS responses**: DNSSEC ensures that DNS responses are authenticated, preventing DNS spoofing attacks.
2. **Verification of TLS certificates**: DANE allows email servers to verify TLS certificates using DNS records, providing an additional layer of security beyond traditional certificate authorities.
3. **Protection against man-in-the-middle attacks**: The combination of DNSSEC and DANE helps protect against sophisticated attacks that could intercept or redirect email traffic.

Forward Email has achieved a perfect score on Internet.nl's mail test, which specifically evaluates

[27]Forward Email Domains Model with Cloudflare DNS Cache Purging: https://github.com/forwardemail/forwardemail.net/blob/master/app/models/domains.js
[28]Cloudflare DNS Cache API documentation: https://developers.cloudflare.com/1.1.1.1/api/
[29]DNSSEC and DANE implementation details: https://internet.nl/mail/forwardemail.net/

DNSSEC and DANE implementation[30].

While DANE is implemented at the DNS level for Forward Email, it is not available in the codebase due to fundamental limitations with Node.js itself. Specifically, TLSA record support is not available in Node.js LTS, therefore underlying packages have not added support for it yet. This limitation is well documented in the Node.js ecosystem[[91]][92]. As noted by the zone-mta developers: "DANE is not planned, mostly because Node.js does not natively support resolving TLSA records." This is further confirmed in the Node.js core issues where developers have requested TLSA record support, noting that "the DNS api don't have a resolveTLSA" function.

This limitation affects not only Forward Email but all Node.js-based email services, as core mail handling libraries like `wildduck` and `mailauth` cannot implement DANE lookups without native TLSA support in the Node.js DNS API. Forward Email continues to monitor this situation and will implement full DANE support in the codebase once the underlying Node.js platform adds the necessary TLSA record resolution capabilities.

7.6 MTA-STS Implementation

In addition to DNSSEC and DANE, Forward Email implements MTA-STS (Mail Transfer Agent Strict Transport Security) to further enhance email security.

MTA-STS provides:

1. **Enforced TLS for email transmission**: MTA-STS ensures that email is only transmitted over secure, encrypted connections.
2. **Protection against downgrade attacks**: By enforcing TLS, MTA-STS prevents attackers from downgrading connections to unencrypted or less secure protocols.
3. **Policy caching**: As shown in the code excerpt, MTA-STS policies are cached with appropriate TTL values, ensuring efficient policy enforcement while maintaining security.

7.7 Transparent DNS Infrastructure

Forward Email's DNS infrastructure draws inspiration from the System Transparency framework outlined in Mullvad's whitepaper[31], which emphasizes the importance of transparent, verifiable systems. Similar to how System Transparency provides verifiable boot processes, Forward Email's DNS infrastructure provides:

1. **Transparent DNS resolution**: All DNS queries and responses can be audited and verified.
2. **Consistent behavior across the infrastructure**: Every server follows the same DNS resolution process, making the system more predictable and secure.

[30]Internet.nl test results: https://internet.nl/mail/forwardemail.net/
[31]Mullvad's System Transparency whitepaper: https://mullvad.net/media/system-transparency-rev4.pdf

3. **Resilience against tampering**: The use of DNSSEC and encrypted DNS over HTTPS prevents manipulation of DNS data.

7.8 EFAIL Protection and Email Security

Forward Email's DNS infrastructure plays a crucial role in protecting against sophisticated email attacks like those described in the EFAIL research[32][33]. By implementing proper DNS security measures, Forward Email helps prevent:

1. **DNS-based exfiltration channels**: Secure DNS prevents attackers from using DNS as a channel to exfiltrate decrypted content.
2. **Man-in-the-middle attacks on email transport**: DNSSEC, DANE, and MTA-STS work together to ensure that email is transmitted securely to the correct destination.
3. **Downgrade attacks on email security**: Proper DNS security prevents attackers from forcing email to be transmitted over insecure channels.

7.9 Advantages Over Traditional DNS Solutions

Forward Email's application-layer approach with Tangerine and DNS over HTTPS provides significant advantages over traditional solutions like unbound or local DNS[34]:

1. **Simplified deployment and maintenance**: No need to configure and maintain separate DNS services on each server, reducing operational complexity.
2. **Enhanced privacy and security**: All DNS queries are encrypted via HTTPS, preventing ISPs or network operators from monitoring or manipulating DNS traffic.
3. **Consistent behavior across environments**: The same DNS resolution logic works identically in development, staging, and production environments.
4. **Improved testability**: The application-layer approach makes it easier to write tests against DNS-related infrastructure, as cache manipulation is much simpler than mocking DNS servers.
5. **Protection against DNS poisoning attacks**: DNS over HTTPS with Cloudflare provides built-in protection against cache poisoning and other DNS-based attacks.

7.10 Performance Optimization

Forward Email's DNS infrastructure includes several performance optimizations:

1. **Timeout and retry configuration**: As shown in the code excerpt, timeouts and retry attempts are configured differently for production and non-production environments:

[32] EFAIL research on email security vulnerabilities: https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-poddebniak.pdf
[33] https://efail.de/
[34] "Why We Built Tangerine": https://forwardemail.net/blog/docs/why-we-built-tangerine-dns-over-https

```
timeout: env.NODE_ENV === 'production' ? 10000 : 5000,
tries: env.NODE_ENV === 'production' ? 4 : 2,
```

2. **Multiple DNS providers**: By using both Cloudflare and Google DNS servers, Forward Email ensures reliability even if one provider experiences issues.

3. **Efficient caching**: The Redis-based caching system with proper TTL handling ensures that DNS lookups are as efficient as possible while maintaining accuracy.

7.11 Conclusion

Forward Email's DNS infrastructure, built around the Tangerine DNS over HTTPS implementation, provides a robust, secure, and high-performance foundation for email services. By implementing DNS at the application layer with proper caching, DNSSEC, DANE, and MTA-STS, Forward Email achieves consistent, secure DNS resolution across all servers, protecting against a wide range of DNS-based attacks and ensuring reliable email delivery.

The approach aligns with principles from the Dark Internet Mail Environment (DIME) architecture[35], which emphasizes the importance of secure, private communication channels starting from the foundational infrastructure layers. By securing the DNS layer, Forward Email establishes a solid foundation for secure email communication.

8 Future Roadmap

Forward Email's commitment to privacy, security, and innovation drives a comprehensive roadmap for future development. This section outlines planned enhancements, new features, and strategic directions that will shape the service's evolution in the coming years, based on actual development plans rather than speculation.

8.1 Core Infrastructure Enhancements

8.1.1 Distributed Architecture Evolution

Forward Email plans to further enhance its distributed architecture to improve resilience and performance:

1. **Multi-Region Redundancy**: Expanding infrastructure across additional geographic regions to enhance availability and provide users with more options for data locality[93].

2. **Improved Load Balancing**: Implementing more sophisticated load balancing algorithms to distribute traffic efficiently across servers.

3. **Enhanced Failover Mechanisms**: Developing more robust failover mechanisms to ensure continuous service availability even during hardware or network failures.

[35] Dark Internet Mail Environment architecture: https://ia803207.us.archive.org/34/items/Dark_Mail_specifications_dark-internet-mail-environment-june-2018/dark-internet-mail-environment-june-2018.pdf

8.1.2 Performance Optimization

Continuous performance improvements are planned through:

1. **SQLite Optimization**: Further optimizing SQLite configurations and query patterns for email-specific workloads, building on the current implementation:

```
const ms = require('ms');
const pWaitFor = require('p-wait-for');

const logger = require('#helpers/logger');

async function closeDatabase(db) {
  if (!db.open) return;
  if (db.inTransaction) {
    try {
      await pWaitFor(() => !db.inTransaction, {
        timeout: ms('30s')
      });
    } catch (err) {
      err.message = `Shutdown could not cancel transaction: ${err.message}`;
      err.isCodeBug = true;
      logger.error(err, { db });
    }
  }

  try {
    db.pragma('analysis_limit=400');
    db.pragma('optimize');
    db.close();
  } catch (err) {
    logger.error(err, { db });
  }
}
```

2. **Redis Cluster Implementation**: Expanding our Redis implementation to support clustering for improved scalability and fault tolerance:

3. **Protocol Optimizations**: Enhancing SMTP, IMAP, and POP3 implementations with custom optimizations for faster message processing and retrieval.

4. **Memory Management Improvements**: Refining in-memory processing techniques to handle larger messages and higher throughput while maintaining security.

8.2 Security Enhancements

8.2.1 Post-Quantum Cryptography Expansion

Building on existing quantum-resistant measures:

1. **NIST PQC Standards Adoption**: Implementing NIST-standardized post-quantum cryptographic algorithms as they become finalized[94].

2. **Hybrid Cryptographic Approaches**: Deploying hybrid classical/post-quantum cryptographic solutions to ensure security during the transition period.

3. **Enhanced ChaCha20-Poly1305 Implementation**: Further optimizing the current ChaCha20-Poly1305 implementation for better performance while maintaining its quantum-resistant properties.

8.2.2 Trusted Platform Module (TPM) Integration

Forward Email is researching the integration of TPM and secure enclave technologies:

1. **Server-Side TPM Utilization**: Leveraging TPM capabilities in server infrastructure to enhance key protection and provide hardware-based security guarantees.
2. **Remote Attestation Framework**: Developing a remote attestation system that allows users to verify the integrity of Forward Email's running software.
3. **Secure Enclaves for Processing**: Exploring the use of technologies like Intel SGX, AMD SEV, or ARM TrustZone for processing emails in protected memory regions.

8.2.3 Advanced Threat Protection

Enhancing security through advanced threat detection and prevention:

1. **Behavioral Analysis**: Implementing behavioral analysis systems to detect anomalous patterns that might indicate compromise.
2. **Zero-Day Vulnerability Protection**: Developing heuristic-based protections against unknown vulnerabilities.
3. **Supply Chain Security**: Enhancing verification of dependencies and implementing reproducible builds for all components, building on Forward Email's existing commitment to transparency[95].

8.3 Privacy Enhancements

8.3.1 Metadata Protection Expansion

Further reducing metadata exposure through:

1. **Enhanced Logger Redaction**: Expanding the current redaction system to cover additional metadata fields:

```
// Current redaction fields definition
const REDACTED_FIELDS = new Set([
  'body',
  'data',
  'password',
  'new_password',
  'pass',
  'passkeys',
  'token',
  'tokens',
  'hash',
  'hashes',
  'salt',
  'tls',
  'ssl',
  'key',
  'cert',
  'ca',
  'dhparam',
  'private_key',
  'dkim_private_key',
  'raw',
  // Additional fields to be added in future updates
]);
```

2. **Enhanced Anonymization**: Developing more sophisticated anonymization techniques for operational data.

3. **Zero-Knowledge Operations**: Expanding zero-knowledge approaches to more aspects of the service.

8.3.2 Authentication Improvements

Enhancing authentication security:

1. **WebAuthn Enhancements**: Expanding the current WebAuthn implementation for stronger passwordless authentication:

```
/**
 * WebAuthn implementation enhancements planned for future releases
 * Building on current implementation:
 */
const {
  SessionChallengeStore
} = require('@forwardemail/passport-fido2-webauthn');

const store = new SessionChallengeStore();
```

2. **Biometric Authentication Integration**: Adding support for biometric authentication in mobile applications while maintaining privacy.

3. **Enhanced Recovery Mechanisms**: Developing more secure account recovery options that maintain the privacy-first approach.

8.4 Feature Expansions

8.4.1 Webmail Service Development

A major focus of Forward Email's roadmap is the development of a comprehensive webmail service[96]:

1. **Modern Web Interface**: Creating a responsive, accessible web interface for email management.

2. **Progressive Web App**: Implementing PWA capabilities for offline access and improved mobile experience.

3. **End-to-End Encryption**: Building end-to-end encryption directly into the webmail interface for seamless secure communication.

4. **Desktop Mail Application**: Developing a native desktop mail application to complement the webmail and mobile offerings. Forward Email is currently in the research and development phase, experimenting with various frameworks including Tauri, Electron, Neutralino, and Flutter to determine the optimal approach for a secure, performant desktop experience.

8.4.2 Enhanced Collaboration Tools

Expanding beyond basic email to provide integrated collaboration features:

1. **Secure Document Sharing**: Implementing end-to-end encrypted document sharing capabilities.
2. **Calendar Enhancements**: Expanding CalDAV functionality with advanced scheduling and sharing features.
3. **Contact Management**: Developing a comprehensive contact management system with privacy-preserving features, including planned support for CardDAV[97]. This will enable seamless synchronization of contacts across devices while maintaining Forward Email's strong privacy guarantees.

8.4.3 Mobile Experience Improvements

Enhancing the mobile experience through:

1. **Native Mobile Applications**: Developing native mobile applications for iOS and Android with enhanced security features. For Android, Forward Email plans to provide multiple distribution channels including Google Play, F-Droid, GitHub Releases, and direct APK downloads with GPG signatures to ensure maximum accessibility and security verification.
2. **Push Notification Privacy**: Implementing private push notification mechanisms that don't expose metadata to third-party services.
3. **Offline Capabilities**: Expanding offline functionality for mobile users.

8.5 Community and Ecosystem Development

8.5.1 Open Source Contribution Strategy

Strengthening the open-source ecosystem:

1. **Contribution Programs**: Establishing formal programs to encourage community contributions.
2. **Educational Resources**: Developing educational materials about email privacy and security.
3. **Research Partnerships**: Forming partnerships with academic institutions for privacy and security research.

8.5.2 Self-Hosting Improvements

Enhancing self-hosting capabilities:

1. **Simplified Deployment**: Creating one-click deployment options for major cloud

providers, building on the current self-hosting documentation[98].

2. **Hardware Appliance Specifications**: Developing specifications for dedicated hardware appliances.
3. **Improved Documentation**: Expanding the comprehensive documentation for self-hosting Forward Email.

8.6 Regulatory and Compliance Roadmap

8.6.1 Compliance Framework Expansion

Preparing for evolving regulatory requirements:

1. **Geographic Expansion**: Ensuring compliance with regulations in additional jurisdictions.
2. **Certification Programs**: Pursuing additional security and privacy certifications beyond the current perfect scores on security tests[99].
3. **Compliance Automation**: Developing tools to automate compliance verification and reporting.

8.7 Implementation Timeline

Forward Email's roadmap implementation follows a phased approach:

8.7.1 Short-Term (6-12 Months)

- Performance optimization enhancements
- Initial webmail service development
- Mobile application development
- API expansion
- Self-hosting improvements

8.7.2 Medium-Term (1-2 Years)

- Post-quantum cryptography expansion
- Advanced threat protection implementation
- Collaboration tools development
- Enhanced authentication methods
- Expanded geographic infrastructure

8.7.3 Long-Term (2-5 Years)

- TPM and secure enclave integration
- Comprehensive webmail platform
- Hardware appliance development

- Advanced privacy-preserving technologies
- Expanded compliance certifications

8.8 Commitment to Open Development

All roadmap items will be developed following Forward Email's commitment to open-source principles:

1. **Public Planning**: Roadmap items will be publicly tracked in GitHub issues and projects
2. **Community Input**: Feature prioritization will include community feedback
3. **Transparent Development**: All code will continue to be developed in the open
4. **Security First**: Security and privacy implications will be primary considerations for all new features

8.9 Conclusion

Forward Email's future roadmap represents a comprehensive vision for advancing email privacy, security, and functionality. By continuing to innovate in areas like quantum-resistant cryptography, authentication security, and webmail development, while maintaining an unwavering commitment to open-source development, Forward Email aims to set new standards for what users should expect from email services.

The planned enhancements will further differentiate Forward Email from competitors by deepening its technical advantages in security architecture, expanding its feature set to address evolving user needs, and strengthening its position as the most transparent and privacy-focused email service available.

As with all aspects of Forward Email, this roadmap itself is open to community input and will evolve based on emerging technologies, security research, and user feedback.

9 Legal Jurisdiction and Government Requests

Forward Email operates under US jurisdiction, which has specific implications for user privacy and security. This section examines the legal framework governing Forward Email's operations, how we handle government requests, and how our approach compares to email providers in other jurisdictions.

9.1 Understanding the CLOUD Act

The Clarifying Lawful Overseas Use of Data (CLOUD) Act was enacted in 2018 to address challenges in accessing electronic information held by service providers for law enforcement purposes. The CLOUD Act has significant implications for email services operating under US jurisdiction[36].

[36]U.S. Department of Justice, "Promoting Public Safety, Privacy, and the Rule of Law Around the World: The Purpose and Impact of the CLOUD Act," April 2019.

9.1.1 Key Provisions of the CLOUD Act

The CLOUD Act clarified that US law requires providers subject to US jurisdiction to disclose data responsive to valid US legal process, regardless of where the company stores the data[37]. This means:

1. US authorities can compel US-based providers to produce data stored on servers anywhere in the world
2. The Act enables executive agreements between the US and foreign governments with robust privacy protections
3. These agreements allow foreign partners to request data directly from US companies under certain conditions

9.1.2 Forward Email's Position Under the CLOUD Act

As a US-based service, Forward Email is subject to the CLOUD Act's provisions. However, our technical architecture provides inherent protections that limit what data could be disclosed in response to legal requests:

1. **Zero-Knowledge Architecture**: Our individually encrypted SQLite mailboxes mean we cannot access the content of users' emails
2. **Minimal Data Collection**: We collect only the minimum information necessary to provide our service
3. **No Logging of Email Content**: We do not log or store email content or metadata to disk
4. **Transparent Privacy Policy**: We clearly communicate what limited data we do collect and how it would be handled in case of legal requests

9.2 Legal Approach to Government Requests

Forward Email is committed to protecting user privacy while complying with valid legal obligations. Our approach to government requests is guided by several key principles:

9.2.1 Handling Law Enforcement Requests

Forward Email has a clear, documented process for handling law enforcement requests as outlined in our Report Abuse documentation[38]. Our ability to disclose information is governed by the Electronic Communications Privacy Act (ECPA), which mandates that we disclose certain user information to law enforcement only in response to specific types of legal requests, including subpoenas, court orders, and search warrants.

For law enforcement requests, we require: 1. Specific account information and date/time ranges rather than overly broad requests 2. Valid legal process (subpoena, ECPA US court order,

[37]The CLOUD Act clarified the U.S. Stored Communications Act to enable the framework envisioned by the CLOUD Act, that each nation would use its own law to access data.
[38]Forward Email, "Report Abuse," https://forwardemail.net/report-abuse

and/or search warrant) 3. Requests from outside the US must come through one of the following channels: - A United States court - An enforcement agency under a United States mutual legal assistance treaty (MLAT) - An order from a foreign government under an executive agreement that satisfies US legal requirements

With the exception of emergencies, we share account information only upon receipt of valid legal process. When legally permitted, we notify users about law enforcement requests unless prohibited by law or court order.

9.2.2 Challenging Overbroad Requests

We carefully review all legal requests and challenge those that are overbroad, vague, or otherwise legally deficient. We draw inspiration from companies like Apple that have taken strong legal positions against government overreach[39].

9.2.3 Emergency Requests

For emergency situations, Forward Email has established a specific process for handling emergency data requests[40]. As permitted under US law (18 U.S.C. §2702(b)(8) and §2702(c)), we may disclose account information to law enforcement without a subpoena, court order, or search warrant when we believe in good faith that doing so without delay is necessary to prevent death or serious physical injury.

For emergency requests, we: 1. Independently verify the email header metadata for authenticity 2. Make good faith attempts to independently contact the requester by phone 3. Require that emergency data requests be sent via email with all relevant information

We are aware of sophisticated spoofing and impersonation attacks and take precautions to verify the legitimacy of all emergency requests.

In the landmark Apple-FBI encryption dispute of 2016, Apple refused an FBI request to create a custom operating system that would disable security features on an iPhone. Apple argued that such a request was unlawful and unconstitutional, as it would undermine the security of all Apple devices and set a dangerous precedent[41].

9.2.4 Legal Basis for Resisting Backdoor Requests

If Forward Email were to receive a request to implement a backdoor or compromise our encryption, we would vigorously fight such requests on multiple legal grounds:

1. **First Amendment Protections**: Code is considered speech, and forcing a company to write code that undermines its security systems could violate First Amendment protec-

[39]Electronic Privacy Information Center, "Apple v. FBI," https://epic.org/documents/apple-v-fbi-2/

[40]Forward Email, "Report Abuse - Emergency Data Requests," https://forwardemail.net/report-abuse#emergency-data-requests

[41]In the Matter of the Search of An Apple iPhone Seized During the Execution of a Search Warrant on a Black Lexus IS300, California License Plate 35KGD203, No. 16-cm-00010 (C.D. Cal. Feb. 16, 2016).

tions, as established in the Bernstein case[42].

2. **All Writs Act Limitations**: The government has previously used the All Writs Act to compel technical assistance, but this authority has limitations. As Apple argued, the Act does not provide a basis to conscript a company to create software enabling the government to hack into encrypted systems[43].

3. **Technical Infeasibility**: Our system is designed so that implementing a backdoor would require a complete architectural redesign, making such requests technically infeasible without destroying the service itself.

9.2.5 The Lavabit Precedent

The case of Lavabit provides an important precedent for email providers facing government demands for encryption keys. In 2013, Lavabit's founder Ladar Levison shut down his secure email service rather than comply with an FBI order to hand over SSL keys that would have compromised all users' privacy[44].

Levison initially attempted to comply by providing the encryption keys as an 11-page printout in 4-point type, arguing that handing over the keys in digital form would compromise the privacy of all 400,000+ Lavabit users. When ordered to provide the keys electronically under threat of criminal contempt, Levison ultimately chose to shut down the service entirely rather than undermine his users' privacy[45].

Forward Email's commitment to user privacy is similarly resolute. While we would exhaust all legal remedies to fight improper requests, we maintain the option to discontinue the service rather than compromise our core privacy principles.

9.3 Transparency in Legal Process

9.3.1 Transparency Reporting

Forward Email is committed to transparency about government requests. We publish regular transparency reports that include:

1. The number and types of government requests received
2. How many requests resulted in disclosure of information
3. The percentage of requests where we challenged or narrowed the scope
4. Any legal constraints on our reporting

[42]Bernstein v. U.S. Department of Justice, 176 F.3d 1132 (9th Cir. 1999), established that code is speech and that restrictions on the dissemination of encryption software burdened First Amendment rights.
[43]Apple's Motion to Vacate Order Compelling Assistance, In the Matter of the Search of An Apple iPhone, No. 16-cm-00010 (C.D. Cal. Feb. 25, 2016).
[44]The Guardian, "Lavabit founder refused FBI order to hand over email encryption keys," October 3, 2013.
[45]The Guardian, "Lavabit loses contempt of court appeal over Edward Snowden encryption keys," April 16, 2014, https://www.theguardian.com/technology/2014/apr/16/lavabit-court-ruling-edward-snowden-encryption

9.3.2 Warrant Canaries as a Transparency Tool

A warrant canary is a statement that declares an organization has not received certain types of legal requests that would come with gag orders prohibiting disclosure[46]. By regularly publishing these statements, services can indirectly inform users if they have received such requests by removing the corresponding canary.

9.3.2.1 How Warrant Canaries Work The concept is similar to the "canary in the coal mine" - when the canary disappears, it signals danger. In the digital context:

1. A service regularly publishes statements that certain legal requests have not been received
2. If such a request is received with a gag order, the service cannot say it received the request
3. However, the service can simply remove the canary statement in its next update
4. Users can infer from the missing statement that such a request was received

9.3.2.2 Legal Basis and Limitations Warrant canaries operate on the legal principle that while the government may be able to compel silence through a gag order, it may not be able to compel an organization to lie by falsely stating it has not received legal process when it has[47]. The First Amendment protects against compelled speech, and courts have rarely upheld compelled false speech.

However, their legal status remains somewhat untested in court. While warrant canaries provide an additional layer of transparency, their legal enforceability continues to be a subject of debate in legal circles.

9.3.2.3 Examples in Practice Several privacy-focused services employ warrant canaries, including:

1. **Cloudflare**: Maintains specific canaries about encryption keys, equipment installation, and content modification[^legal16d]. Their canaries include statements such as:
    - "Cloudflare has never turned over our encryption or authentication keys or our customers' encryption or authentication keys to anyone."
    - "Cloudflare has never installed any law enforcement software or equipment anywhere on our network."
    - "Cloudflare has never provided any law enforcement organization a feed of our customers' content transiting our network."
2. **Apple**: Has previously used canaries regarding national security letters, stating "Apple has never received an order under Section 215 of the USA Patriot Act."[^legal16e]

[46] Cloudflare, "What is a warrant canary?", https://www.cloudflare.com/learning/privacy/what-is-warrant-canary/

[47] Electronic Frontier Foundation, "Warrant Canary Frequently Asked Questions", https://www.eff.org/deeplinks/2014/04/warrant-canary-faq "Lavabit founder refused FBI order to hand over email encryption keys," October 3, 2013, https://www.theguardian.com/world/2013/oct/03/lavabit-ladar-levison-fbi-encryption-keys-snowden

3. **Various VPN providers**: Many include canaries in their transparency reports, with some publishing them as frequently as daily or weekly

9.3.2.4 Effectiveness and Considerations While warrant canaries can provide an additional layer of transparency, they have limitations:

1. They only indicate that some type of request was received, not specifics about the request
2. Their legal enforceability remains uncertain
3. They require vigilant monitoring by users to be effective
4. Some services use "binary" canaries (present/absent) while others use more detailed statements
5. The timing of updates can be critical - too frequent may reduce their effectiveness, too infrequent may delay notification

Despite these limitations, warrant canaries remain an important tool in the privacy ecosystem, allowing services to communicate with users about government requests in ways that might otherwise be prohibited.

9.3.3 User Notification

When legally permitted, we notify users about government requests for their information. This notification policy includes:

1. Advance notice when possible to allow users to challenge requests
2. Post-disclosure notice when advance notice is prohibited
3. Fighting gag orders that prevent user notification

9.4 Limited Data Available for Legal Requests

In the event of a valid legal subpoena that withstands all challenges, the information Forward Email could provide is extremely limited:

1. **Account Information**: Basic subscriber information such as email address, signup date, and payment information (if applicable)
2. **Connection Logs**: Limited IP address logs that may be temporarily stored for security and abuse prevention
3. **No Email Content**: Due to our zero-knowledge architecture, we cannot access the encrypted content of emails
4. **No Metadata Analysis**: We do not analyze or store metadata about who users communicate with

This limited data collection approach is a deliberate technical and policy decision that protects user privacy even in the face of valid legal process.

9.5 Jurisdictional Comparison with Other Privacy-Focused Services

9.5.1 Understanding Surveillance Alliances

Before comparing specific jurisdictions, it's important to understand the global surveillance alliances that affect privacy services worldwide. These intelligence-sharing agreements create frameworks for coordinated intelligence gathering across borders[48].

9.5.1.1 Five Eyes, Nine Eyes, and Fourteen Eyes The **Five Eyes** alliance originated from the UKUSA Agreement, a multilateral agreement for cooperation in signals intelligence between Australia, Canada, New Zealand, the United Kingdom, and the United States[49]. This alliance allows member countries to share intelligence and conduct coordinated surveillance activities.

The **Nine Eyes** expands this alliance to include Denmark, France, the Netherlands, and Norway, while the **Fourteen Eyes** (officially called SIGINT Seniors Europe or SSEUR) further includes Belgium, Germany, Italy, Spain, and Sweden.

These alliances are significant for privacy services because: 1. Intelligence agencies within these alliances share data with each other 2. While agencies typically cannot spy on their own citizens, they can receive such information from partner agencies 3. Services operating in these jurisdictions may face coordinated legal pressure from multiple governments

9.5.2 Switzerland: Proton Mail

Proton Mail operates under Swiss jurisdiction, which offers certain advantages:

1. **Strong Privacy Laws**: Switzerland has some of the world's strongest privacy protections, including Article 13 of the Swiss Federal Constitution[50]
2. **Outside Fourteen Eyes**: Switzerland is not part of the "Fourteen Eyes" intelligence-sharing alliance
3. **Judicial Process**: Government requests require judicial approval and must meet a higher threshold of evidence
4. **Limitations**: Despite these protections, Proton Mail has complied with thousands of data requests when legally required, providing IP addresses and basic account information[51]

9.5.3 Germany: Tutanota

Tutanota is based in Germany, which presents a different legal environment:

1. **EU Data Protection**: Subject to GDPR, providing strong data protection rights
2. **Fourteen Eyes Member**: Germany is part of the "Fourteen Eyes" intelligence-sharing alliance

[48] ProtonVPN, "What is the Five Eyes alliance?", https://protonvpn.com/blog/5-eyes-global-surveillance
[49] Wikipedia, "UKUSA Agreement," https://en.wikipedia.org/wiki/UKUSA_Agreement
[50] Proton Mail, "Proton Mail vs Tutanota," https://proton.me/mail/proton-mail-vs-tutanota
[51] CyberInsider, "ProtonMail Complied with 5957 Data Requests in 2022," August 15, 2023.

3. **Court Rulings**: German courts have ordered Tutanota to implement monitoring capabilities for specific accounts under investigation[52]. According to Tutanota's transparency report, they regularly release real-time content data and stored encrypted content data in response to German court orders[53]. In 2020, a regional court in Cologne forced Tutanota to develop a function that allows monitoring of an individual account's incoming and outgoing emails[54].
4. **Encryption Limitations**: German authorities have proposed legislation that could require backdoors in encrypted communications. A German judge can either issue a seizure of a mailbox or a real-time monitoring of the mailbox, which Tutanota must comply with under German law[55].

9.5.4 Sweden: Mullvad VPN

While not an email provider, Mullvad VPN's approach to privacy in Sweden offers relevant comparisons:

1. **No-Logs Policy**: Mullvad maintains a strict no-logs policy, collecting no data about users' activities or identities
2. **Cash Payments**: Accepts anonymous cash payments to avoid payment tracking
3. **Swedish Law**: Sweden has strong privacy protections but is a Fourteen Eyes member
4. **Multi-layered Defense Strategy**: Mullvad employs a comprehensive approach to handling government requests[56]:
    - They collect no data about users' activities, making it impossible to fulfill data requests
    - When governments request data, they refer them to their policy and explain they have no information
    - If legally forced to spy on users, they would cease operations in the affected jurisdiction
5. **Swedish Legal Framework**: Mullvad operates under Swedish legislation which provides certain protections[57]:
    - The Electronic Communications Act does not apply to VPN services
    - Swedish law does not allow any government to force them to spy on users
    - They retain lawyers to monitor the legal landscape and stay informed of developments

[52]Tutanota was ordered by a German court to implement a function to deliver unencrypted copies of emails for specific accounts under criminal investigation.
[53]Tuta (formerly Tutanota), "Transparency Report & Warrant Canary," https://tuta.com/blog/transparency-report
[54]TechCrunch, "German secure email provider Tutanota forced to monitor an account, after regional court ruling," December 8, 2020, https://techcrunch.com/2020/12/08/german-secure-email-provider-tutanota-forced-to-monitor-an-account-after-regional-court-ruling/
[55]Tuta (formerly Tutanota), "Transparency Report & Warrant Canary," https://tuta.com/blog/transparency-report#:~:text=A%20German%20judge%20can%20either%20issue%20a%20seizure%20of%20a%20mailbox%20or%20a%20real%20
[56]Mullvad VPN, "How we handle government requests for user data," https://mullvad.net/en/help/how-we-handle-government-requests-user-data
[57]Mullvad VPN, "Swedish legislation," https://mullvad.net/en/help/swedish-legislation

9.5.5 United States: Signal

Signal, while primarily a messaging app rather than an email service, provides important insights into how privacy-focused services respond to legal challenges:

1. **End-to-End Encryption**: Signal uses strong end-to-end encryption for all communications
2. **Minimal Data Collection**: Signal collects virtually no user data, making compliance with data requests technically impossible
3. **Stance on Backdoors**: Signal has taken an uncompromising position against encryption backdoors[58]:
   - Signal's president Meredith Whittaker stated: "Signal's position on this is very clear – we will not walk back, adulterate, or otherwise perturb the robust privacy and security guarantees that people depend on"[100]
   - When faced with potential legal requirements to weaken encryption in the UK and Sweden, Signal made clear it would exit those markets
4. **Response to UK's Online Safety Bill**: Signal threatened to leave the UK market rather than comply with requirements that would undermine encryption[59]:
   - The bill would require platforms to scan for prohibited content, even in encrypted communications
   - Signal argued this would be technically impossible without breaking encryption for all users
5. **Response to Sweden's Encryption Backdoor Law**: Signal took a similar position regarding Sweden's proposed legislation[60]:
   - Whittaker stated: "In practice, this means that we are asked to break the encryption that is the foundation of our entire operation. Asking us to store data would undermine our entire architecture, and we would never do that. We would rather leave the Swedish market entirely,"[[101]][100]
   - Signal emphasized that backdoors cannot be limited to specific jurisdictions or purposes
6. **Technical Reality**: Signal has emphasized that encryption backdoors are not just policy choices but technical impossibilities if security is to be maintained[61]:
   - Creating backdoors would undermine security for all users globally
   - Once backdoors exist, they can potentially be exploited by malicious actors

[58] TechRadar, "We will not walk back: Signal would rather leave the UK and Sweden than remove encryption protections", https://www.techradar.com/computing/cyber-security/we-will-not-walk-back-signal-would-rather-leave-the-uk-and-sweden-than-remove-encryption-protections

[59] BBC News, "Signal threatens to quit UK over Online Safety Bill", https://www.bbc.com/news/technology-64584001

[60] Sweden Herald, "Signal's CEO: Then We're Leaving Sweden", https://swedenherald.com/article/signals-ceo-then-were-leaving-sweden

[61] Signal, "There is no such thing as a backdoor that only the good guys can access", https://signal.org/blog/there-is-no-backdoor/

9.6 Forward Email's Jurisdictional Advantages

While operating under US jurisdiction presents certain challenges, Forward Email maintains several advantages:

1. **Constitutional Protections**: US constitutional protections, including the First and Fourth Amendments, provide strong legal grounds to challenge overreaching government requests
2. **Legal Precedents**: Established case law in the US provides a framework for challenging improper government demands
3. **Technical Design**: Our zero-knowledge architecture means we simply cannot access the content of emails, regardless of legal jurisdiction
4. **Transparency**: US law allows for more detailed transparency reporting than some other jurisdictions
5. **Jurisdictional Stability**: Unlike some jurisdictions with rapidly changing privacy laws, the US legal system provides relative stability and predictability

9.7 Single Jurisdiction Benefits

Operating solely within US jurisdiction provides important benefits compared to services that operate across multiple jurisdictions:

1. **Legal Clarity**: Users know exactly which legal framework applies to their data
2. **Avoiding Jurisdictional Conflicts**: Services operating in multiple countries may face conflicting legal requirements
3. **Preventing Forum Shopping**: Government agencies cannot "shop" for the most favorable jurisdiction to request data
4. **Consistent Legal Defense**: Allows for consistent legal strategies when challenging improper requests

9.8 Conclusion

Forward Email operates under US jurisdiction with a clear understanding of the legal implications for user privacy. Through our technical architecture, minimal data collection practices, and commitment to legal advocacy, we provide strong privacy protections despite jurisdictional challenges. We remain committed to transparency about government requests and will vigorously defend user privacy through all available legal means.

Our approach demonstrates that strong privacy protections are possible under US jurisdiction when combined with the right technical architecture and legal stance. By understanding the legal landscape and designing our systems accordingly, we provide users with a secure email service that respects their privacy while operating within the bounds of applicable law.

10 Acknowledgements

We would like to express our sincere gratitude to the following individuals and organizations who have contributed to Forward Email's development and success:

- Joseph Jacks (OSS Capital) – our main advisor; endless encouragement, criticism, and conviction
- Andris Reinman (Nodemailer, WildDuck) – prolific open-source author (we use at least ten of their libraries including `libmime`, `mailauth`, `mailparser`, `mailsplit`, `mobileconfig`, `mx-connect`, `nodemailer`, `smtp-server`, `wildduck`, `zone-mta`)
- Sindre Sorhus – prolific open-source author (we use 46+ npm packages they created)
- Shaun Warman – contributor that integrated 2FA, OTP, CSP, SRI, and more into Lad, which is the original framework powering Forward Email; also single handedly created the self-hostable Forward Email, and made it possible for users to self-host Forward Email in 1-click (Self-Hosted Solution)
- Scott Kitterman (Author of SPF, RFC 7208) – helped over IRC in early days when we had Python spawned child process scripts using `dkimpy`) (see 1 and 2
- Fedor Indutny – prolific Node.js contributor that helped with cipher ordering in early days
- Node.js and JavaScript Community
- Spencer Snyder – contributor
- Taylor Schley – contributor
- Elmer Melendez – contributor
- Paul Graham – pioneering work on email spam filtering
- Ryan Sipes (Thunderbird) – contributor and industry peer; we thank them for encouragement and conviction over the years
- Paul Irish – for making the web fun decades ago and for HTML5 Boilerplate
- Tom Preston-Werner – for creating GitHub, SemVer, and README-first approach
- Isaac Z. Schlueter – for creating npm and contributing to Node.js
- Ryan Dahl – for creating Node.js
- TJ Holowaychuk – for being so prolific, creating many packages we use, and building Express
- Imed Jaberi – for helping maintain packages in the Koa and Express ecosystems
- Express Technical Committee
- Node.js Technical Steering Committee

- Linux Foundation
- Linus Torvalds
- Peter Cooper – for supporting our OSS efforts in Cooper Press (JavaScript Weekly and Node Weekly)
- Mahesh Bandara Wijerathna – for creating better-sqlite3-multiple-ciphers
- Eugene Lazutkin – for creating re2
- Joshua Wise – for creating better-sqlite3
- The authors, maintainers, and contributors of 392+ packages we use

References

[1] F. E. Team, "Forward email." 2025. Available: https://forwardemail.net

[2] W. Database, "Forward email domain registration." 2017. Available: https://whois.domaintools.com/forwardemail.net

[3] B. Butler, "Proton mail backend code discussion." 2023. Available: https://github.com/ProtonMail/WebClients/issues/257#issuecomment-964240013

[4] F. E. Team, "Why open source matters for security." 2024. Available: https://forwardemail.net/blog/why-open-source-matters

[5] D. J. Bernstein, "ChaCha20-Poly1305 and post-quantum security." 2023. Available: https://cr.yp.to/chacha.html

[6] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols." 2018. Available: https://tools.ietf.org/html/rfc8439

[7] S. Team, "SQLite encryption extension." 2024. Available: https://www.sqlite.org/see/doc/trunk/www/readme.wiki

[8] S. Team, "SQLite PRAGMA statements." 2024. Available: https://www.sqlite.org/pragma.html

[9] A. W. Services, "Amazon simple email service (SES)." 2024. Available: https://aws.amazon.com/ses/

[10] F. E. Team, "University of maryland email forwarding case study." 2024. Available: https://forwardemail.net/blog/docs/university-of-maryland-case-study

[11] F. E. Team, "Linux foundation email enterprise case study." 2024. Available: https://forwardemail.net/blog/docs/linux-foundation-email-enterprise-case-study

[12] F. E. Team, "Canonical/ubuntu email enterprise case study." 2024. Available: https://forwardemail.net/blog/docs/canonical-ubuntu-email-enterprise-case-study

[13] F. E. Team, "Forward email security test results." 2025. Available: https://forwardemail.net/security

[14] F. E. Team, "How npm packages with a billion downloads shaped the JavaScript ecosystem." 2024. Available: https://forwardemail.net/blog/docs/how-npm-packages-billion-downloads-shaped-javascript-ecosystem

[15] F. E. Team, "Bree: Job scheduler for node.js." 2024. Available: https://github.com/breejs/bree

[16] F. E. Team, "Cabin: Structured logging system." 2024. Available: https://github.com/cabinjs/cabin

[17] F. E. Team, "Spam scanner: Email spam detection tools." 2024. Available: https://github.com/forwardemail/spam-scanner

[18] F. E. Team, "Tangerine: DNS over HTTPS implementation." 2024. Available: https://www.npmjs.com/package/tangerine

[19] Reddit, "Discussion on tutanota's closed-source backend." 2023. Available: https://old.reddit.com/r/tutanota/comments/10hghin/tutanota_opens_backend_server_side/

[20] I. E. T. Force, “Simple mail transfer protocol.” 2008. Available: https://tools.ietf.org/html/rfc5321

[21] I. E. T. Force, “Internet message access protocol - version 4rev1.” 2003. Available: https://tools.ietf.org/html/rfc3501

[22] I. E. T. Force, “Post office protocol - version 3.” 1996. Available: https://tools.ietf.org/html/rfc1939

[23] Microsoft, “The STRIDE threat model.” 2023. Available: https://learn.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)

[24] T. UcedaVelez and M. M. Morana, “Process for attack simulation and threat analysis (PASTA).” 2015. Available: https://owasp.org/www-pdf-archive/AppSecEU2012_PASTA.pdf

[25] A. Shostack, “Threat modeling: Designing for security.” 2014. Available: https://www.wiley.com/en-us/Threat+Modeling%3A+Designing+for+Security-p-9781118809990

[26] M. VPN, “Mullvad VPN security approach.” 2024. Available: https://mullvad.net/en/help/swedish-legislation

[27] D. J. Bernstein, “ChaCha20-Poly1305 and post-quantum security.” 2023. Available: https://cr.yp.to/chacha.html

[28] F. E. Team, “Forward email self-hosting documentation.” 2024. Available: https://forwardemail.net/blog/docs/self-hosted-solution

[29] O. Foundation, “TOTP vs SMS-based 2FA security analysis.” 2023. Available: https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html

[30] DataPacket, “DataPacket DDoS protection.” 2024. Available: https://datapacket.com/ddos-protection

[31] NIST, “Post-quantum cryptography: ChaCha20-Poly1305.” 2023. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography

[32] I. Society, “DNSSEC and DANE: Securing DNS.” 2023. Available: https://www.internetsociety.org/deploy360/dnssec/

[33] Lavabit, “Dark internet mail environment (DIME) whitepaper.” 2015. Available: https://darkmail.info/downloads/dark-internet-mail-environment-march-2015.pdf

[34] S. T. Project, “System transparency framework.” 2023. Available: https://system-transparency.org/

[35] S. T. Project, “System transparency principles.” 2023. Available: https://system-transparency.org/overview/

[36] Internet.nl, “Internet.nl test results for forwardemail.net.” 2025. Available: https://internet.nl/site/forwardemail.net/

[37] Q. S. Labs, “SSL labs test results for forwardemail.net.” 2025. Available: https://www.ssllabs.com/ssltest/analyze.html?d=forwardemail.net&latest

[38] Mozilla, “Mozilla observatory results for forwardemail.net.” 2025. Available: https://observatory.mozilla.org/analyze/forwardemail.net

[39] Hardenize, "Hardenize security report for forwardemail.net." 2025. Available: https://www.hardenize.com/report/forwardemail.net

[40] OWASP, "Security through obscurity." 2023. Available: https://owasp.org/www-community/Security_through_obscurity

[41] A. Kerckhoffs, "Kerckhoffs's principle." 1883. Available: https://en.wikipedia.org/wiki/Kerckhoffs%27s_principle

[42] F. E. Team, "A decade of impact: How our npm packages hit 1 billion downloads and shaped JavaScript." 2025. Available: https://forwardemail.net/en/blog/docs/how-npm-packages-billion-downloads-shaped-javascript-ecosystem#adherence-to-time-tested-software-development-principles

[43] T. Reenskaug, "Model-view-controller." 1979. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[44] D. McIlroy, "The unix philosophy." 1978. Available: https://en.wikipedia.org/wiki/Unix_philosophy

[45] K. Johnson, "Keep it simple and straightforward (KISS)." 1960. Available: https://en.wikipedia.org/wiki/KISS_principle

[46] A. Hunt and D. Thomas, "Don't repeat yourself (DRY)." 1999. Available: https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

[47] R. Jeffries, "You aren't gonna need it (YAGNI)." 1999. Available: https://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it

[48] A. Wiggins, "The twelve-factor app." 2011. Available: https://12factor.net/

[49] W. of Ockham, "Occam's razor." 1323. Available: https://en.wikipedia.org/wiki/Occam%27s_razor

[50] Microsoft, "Eating your own dog food." 1988. Available: https://en.wikipedia.org/wiki/Eating_your_own_dog_food

[51] P. Graham, "Ramen profitable." 2009. Available: https://www.paulgraham.com/ramenprofitable.html

[52] F. E. Team, "Forward email isCodeBug helper implementation." 2025. Available: https://github.com/forwardemail/forwardemail.net/blob/master/helpers/refine-and-log-error.js

[53] F. E. Team, "Forward email internationalization support." 2025. Available: https://github.com/forwardemail/forwardemail.net/tree/master/locales

[54] F. E. Team, "Mandarin: Node.js internationalization library." 2025. Available: https://github.com/ladjs/mandarin

[55] F. E. Team, "Forward email DNS provider setup guides." 2025. Available: https://forwardemail.net/guides

[56] F. E. Team, "Forward email DNS provider configuration source code." 2025. Available: https://github.com/forwardemail/forwardemail.net/blob/e35b36f5764d6fc3fcdfea937899b85c4e6d6756/config/utilities.js#L71-L354

[57] F. E. Team, "DNS-over-HTTPS implementation." 2025. Available: https://github.com/forwardemail/forwardemail.net/blob/master/helpers/create-tangerine.js

[58] F. E. Team, "Software development principles." 2025. Available: https://forwardemail.net/blog/docs/software-development-principles

[59] F. E. Team, "Targeting developers as early adopters." 2025. Available: https://forwardemail.net/blog/docs/targeting-developers-as-early-adopters

[60] F. E. Team, "Software development principles in practice." 2025. Available: https://forwardemail.net/blog/docs/principles-in-practice

[61] M. Foundation, "Mozilla public license 2.0." 2012. Available: https://www.mozilla.org/en-US/MPL/2.0/

[62] MariaDB, "Business source license 1.1." 2017. Available: https://mariadb.com/bsl11/

[63] O. S. Initiative, "License compatibility analysis." 2023. Available: https://opensource.org/licenses/

[64] TechCrunch, "Jack dorsey's block is building a bitcoin mining system." 2022. Available: https://techcrunch.com/2022/01/13/jack-dorseys-block-is-building-a-bitcoin-mining-system/

[65] hstspreload.org, "HSTS preload list submission." 2025. Available: https://hstspreload.org

[66] Q. S. Labs, "SSL labs server test for forwardemail.net." 2025. Available: https://www.ssllabs.com/ssltest/analyze.html?d=forwardemail.net&latest

[67] Node. js Foundation, "Node.js crypto documentation - PBKDF2." 2024. Available: https://nodejs.org/api/crypto.html#cryptopbkdf2syncpassword-salt-iterations-keylen-digest

[68] O. Foundation, "OWASP password storage cheat sheet." 2024. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[69] zxcvbn-ts Team, "Zxcvbn-ts: Realistic password strength estimation." 2024. Available: https://github.com/zxcvbn-ts/zxcvbn

[70] utelle, "SQLite3MultipleCiphers issue #136." 2023. Available: https://github.com/utelle/SQLite3MultipleCiphers/issues/136

[71] PostQuantum, "Post-quantum cryptography." 2024. Available: https://postquantum.com/post-quantum/quantum-safe-secure-cryptography/

[72] Wikipedia, "SIM swap scam." 2025. Available: https://en.wikipedia.org/wiki/SIM_swap_scam

[73] Reuters, "U.s. SEC's x account hacked with SIM swapping, agency says." 2024. Available: https://www.reuters.com/technology/cybersecurity/us-secs-x-account-hacked-with-sim-swapping-agency-says-2024-01-22/

[74] Bloomberg, "FTX's missing $400 million stolen in SIM-swapping hack, DOJ says." 2024. Available: https://www.bloomberg.com/news/articles/2024-02-01/ftx-s-missing-400-million-stolen-in-sim-swapping-hack-doj-says

[75] T. Verge, “FCC adopts new rules to protect consumers from SIM swapping and port-out fraud.” 2023. Available: https://www.theverge.com/2023/7/11/23791183/fcc-sim-swapping-port-out-phone-hijacking-security-protection

[76] T. Register, “Scattered spider SIM-swap gang ordered to repay $13.2M to victims.” 2025. Available: https://www.theregister.com/2025/04/07/scattered_spider_sim_swap/

[77] H. News, “Discussion on SIM swapping vulnerabilities.” 2023. Available: https://news.ycombinator.com/item?id=37170414

[78] F. E. Team, “Forward email session management implementation.” 2025. Available: https://github.com/forwardemail/forwardemail.net/blob/master/jobs/session-management.js

[79] F. E. Team, “Forward email session invalidation implementation.” 2025. Available: https://github.com/forwardemail/forwardemail.net/blob/master/helpers/invalidate-other-sessions.js

[80] S. Helme, “CSRF is dead.” 2023. Available: https://scotthelme.co.uk/csrf-is-dead/

[81] C. for Internet Security, “Server hardening guide.” 2024. Available: https://www.cisecurity.org/cis-benchmarks/

[82] M3AAWG, “DKIM key sizes.” 2023. Available: https://www.m3aawg.org/sites/default/files/m3aawg-dkim-key-rotation-bp-2019-03.pdf

[83] F. E. Team, “Forward email phishing detection implementation.” 2025. Available: https://github.com/forwardemail/forwardemail.net/blob/e35b36f5764d6fc3fcdfea937899b85c4e6d6756/helpers/is-arbitrary.js#L281-L313

[84] F. E. Team, “Forward email phishing notification system.” 2025. Available: https://github.com/forwardemail/forwardemail.net/blob/e35b36f5764d6fc3fcdfea937899b85c4e6d6756/helpers/on-data-mx.js#L1901-L1912

[85] F. E. Team, “iOS push notification support.” 2023. Available: https://github.com/zone-eu/wildduck/issues/711

[86] P. Guides, “Mailbox.org no encryption for sent email.” 2023. Available: https://discuss.privacyguides.net/t/mailbox-org-no-encryption-for-sent-email/16563

[87] P. Guides, “Remove mailbox.org.” 2023. Available: https://discuss.privacyguides.net/t/remove-mailbox-org/20232/45

[88] Mailbox.org, “Anti-spoofing for custom domains (SPF, DKIM, DMARC).” 2023. Available: https://userforum-en.mailbox.org/topic/anti-spoofing-for-custom-domains-spf-dkim-dmarc#comment-2529

[89] Tarnkappe.info, “Mailbox.org came after the snowden revelations - a talk with peer heinlein.” 2023. Available: https://tarnkappe.info/artikel/interviews/mailbox-org-came-after-the-snowden-revelations-a-talk-with-peer-heinlein-94431.html

[90] Mailbox.org, “Security at mailbox.org.” 2024. Available: https://mailbox.org/en/security

[91] zone-eu, “DANE / MTA-STS support.” 2020. Available: https://github.com/zone-eu/zone-mta/issues/253

[92] nodejs, "[DNS] TLSA records [HTTPS] DANE request." 2021. Available: https://github.com/nodejs/node/issues/39569

[93] I. Society, "The future of email encryption." 2024. Available: https://www.internetsociety.org/resources/doc/2018/email-security/

[94] NIST, "Post-quantum cryptography standardization." 2024. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography

[95] M3AAWG, "Email authentication best practices." 2023. Available: https://www.m3aawg.org/sites/default/files/m3aawg-email-authentication-recommended-best-practices-2023-02.pdf

[96] IETF, "The future of SMTP security." 2024. Available: https://datatracker.ietf.org/doc/html/draft-ietf-uta-email-deep

[97] F. E. Team, "CardDAV support discussion." 2024. Available: https://github.com/orgs/forwardemail/discussions/274

[98] IETF, "MTA-STS: Strict transport security for mail transfer agents." 2018. Available: https://tools.ietf.org/html/rfc8461

[99] IETF, "DANE: DNS-based authentication of named entities." 2012. Available: https://tools.ietf.org/html/rfc6698

[100] L. Mearian, "Signal will exit sweden rather than dilute message security." 2024. Available: https://www.computerworld.com/article/3833959/signal-will-exit-sweden-rather-than-dilute-message-security.html

[101] SVT, "Signal lämnar sverige om regeringens förslag på datalagring klubbas." 2024. Available: https://www.svt.se/nyheter/inrikes/signal-lamnar-sverige-om-regeringens-forslag-pa-datalagring-klubbas