ドメイン認証をメールの受信側に普及させるためのプログラムの設計と実装

IIJ-II
山本和彦

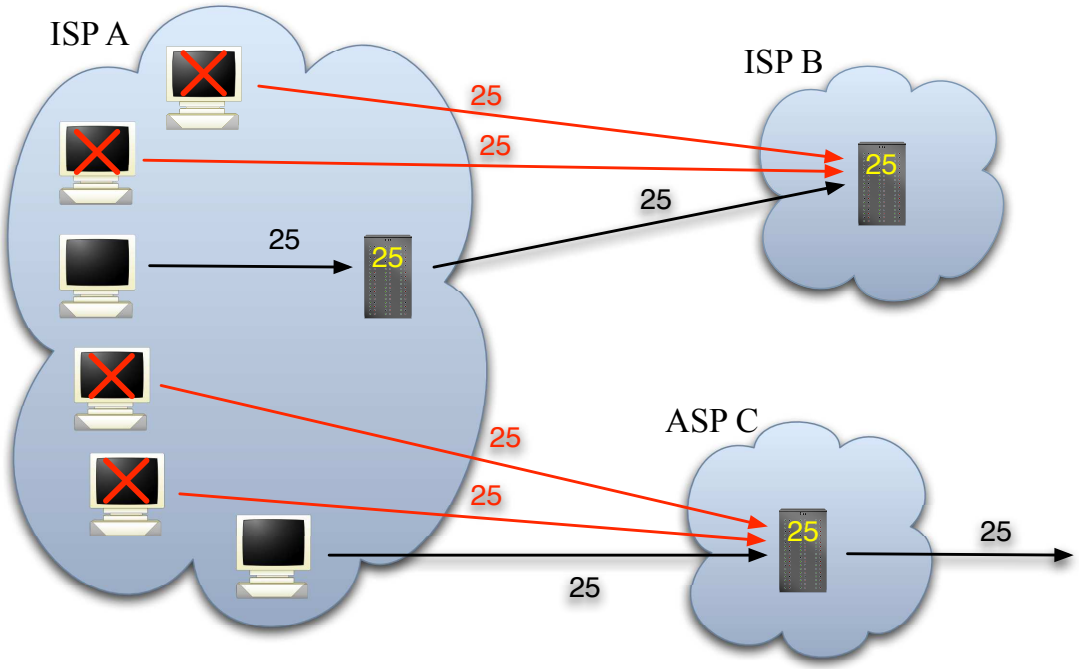
内容

- 迷惑メール対策のロードマップ
- 問題点：ドメイン認証を受信側へ普及させること
- 解決のアイディアと研究の目的
- プログラムの設計
- 実装の特徴的な点
- 4種類のテスト
- ソフトウェアの公開

迷惑メール対策の
ロードマップ

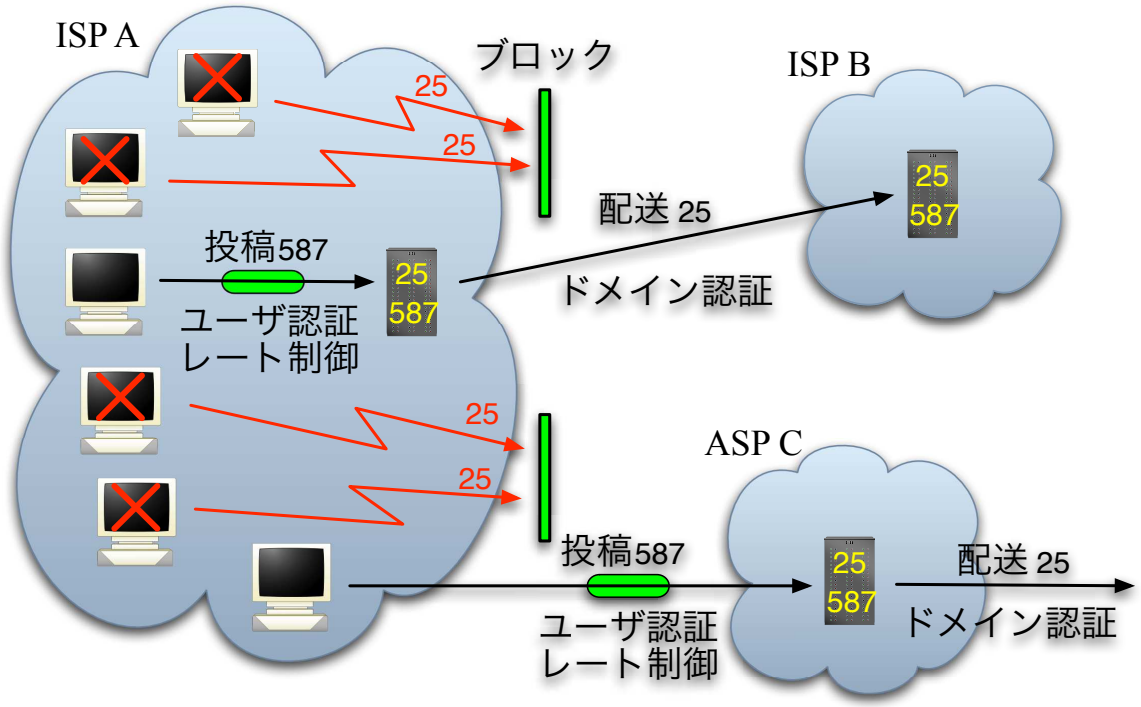
2005年頃の状況

- 大量の迷惑メール
  - BOT や「渡り」



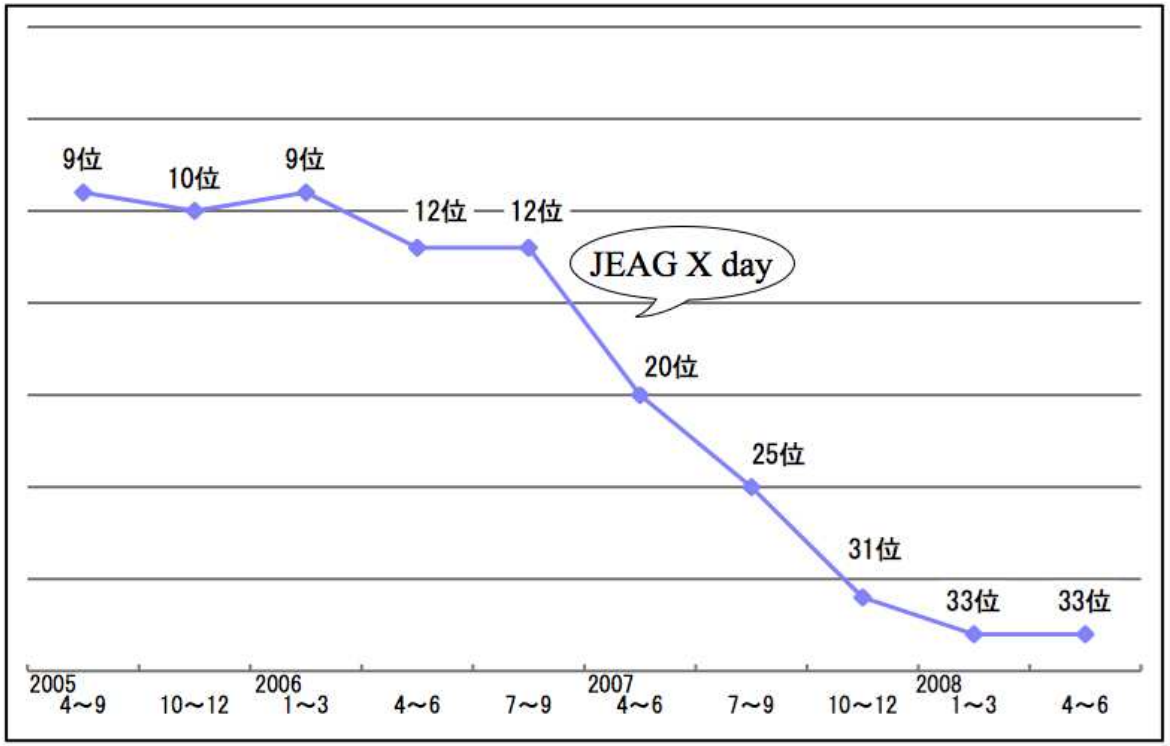
迷惑メール対策のロードマップ

- 25番ポートのブロック、ユーザー認証、ドメイン認証
  - MAAWG(Messaging Anti-Abuse Working Group)
  - JEAG(Japan Email Anti-Abuse Group)



25番ポートのブロックの効果

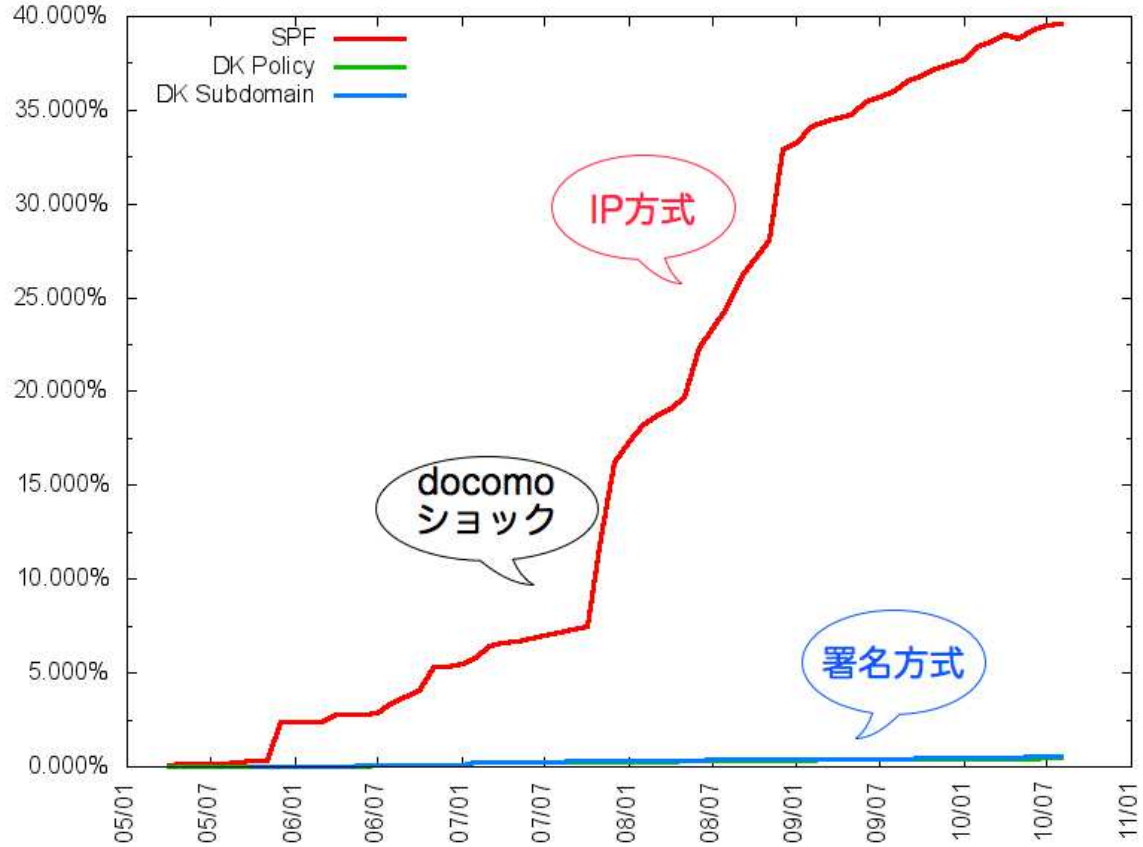
- 迷惑メール送信国のランキング(SOPHOS 社)



- JEAG 勧告
  - ブリッキングを2006年12月までに実施すべきである

ドメイン認証技術の送信側への普及率

- WIDE & JPRS の共同研究
  - .JP 以下のドメイン



迷惑メール対策の
現時点での課題

現時点での課題


25番ポートのブロック


投稿と配送の分離


パスワードによるユーザー認証


ドメイン認証 (送信側)


ドメイン認証 (受信側)

受信側へ普及させる際の問題点

- 正当なメールであっても、ドメインが詐称されていると判断されることがある

IP方式
SPF(Sender Policy Framework)
SenderID

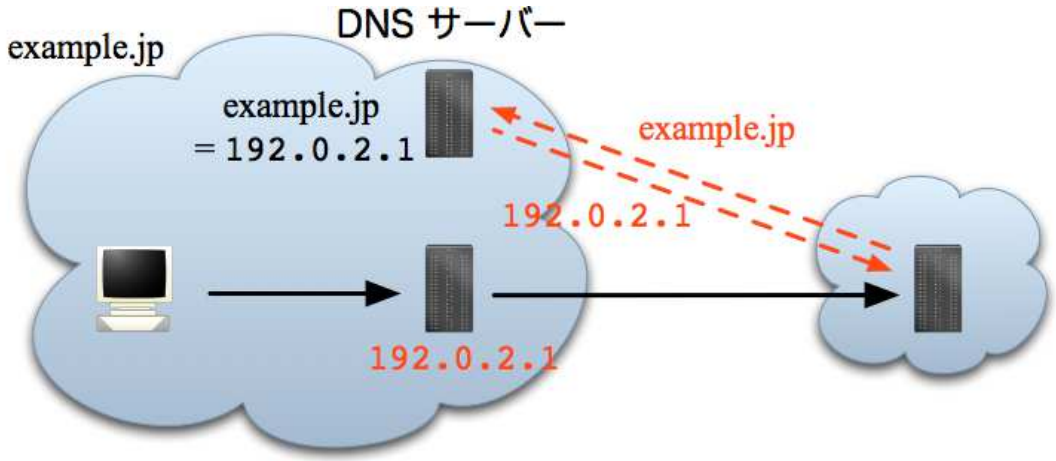
メールの転送

署名方式
DomainKeys
DKIM(DomainKeys Identified Mail)

メーリングリスト

IP 方式のドメイン認証

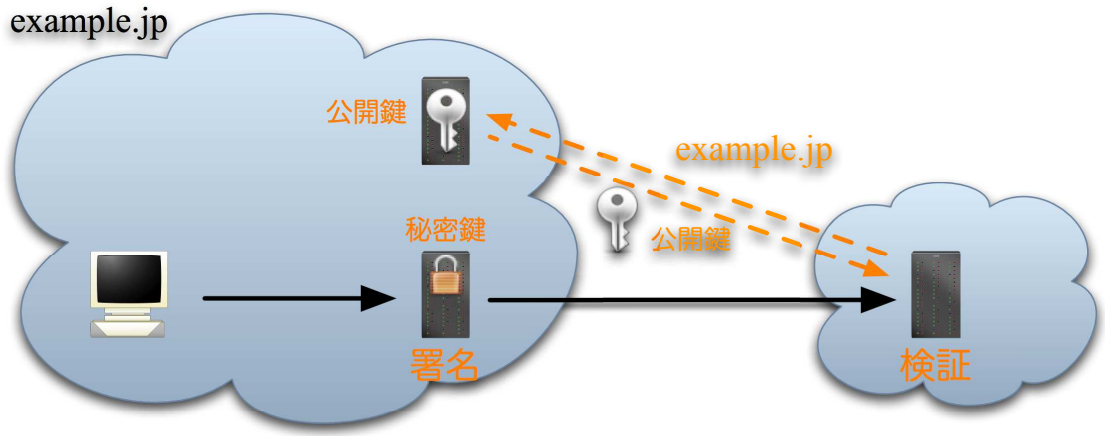
- 送信側は、送信サーバの IP アドレスを DNS で宣言
- 受信側は、送信元の IP アドレスと DNS から得た IP アドレスを比較



- 転送されて送信元の IP アドレスが変わると、ドメインが詐称されているとみなされる

署名方式のドメイン認証

- 送信側は、DNS に公開鍵を登録
- 送信側は、送信サーバでメールに署名
- 受信側は、DNS から得られた公開鍵で署名を検証



- メールが書き換えられると、
ドメインが詐称されているとみなされる
  - メーリングリストでの件名の書き換え

解決のアイディアと
研究の目的

解決のアイディア

Meng Wong 氏

IP 方式と署名方式を組み合わせる

IP方式	署名方式
☠ メールの転送	メールの転送
メーリングリスト	☠ メーリングリスト

研究の目的

ドメイン認証技術を受信側へ
普及させるための一方法として
Meng Wong 氏のアイディアを
実現する

プログラムの設計

設計方針

Milter

SMTP サーバーに依存しない
Sendmail, postfix

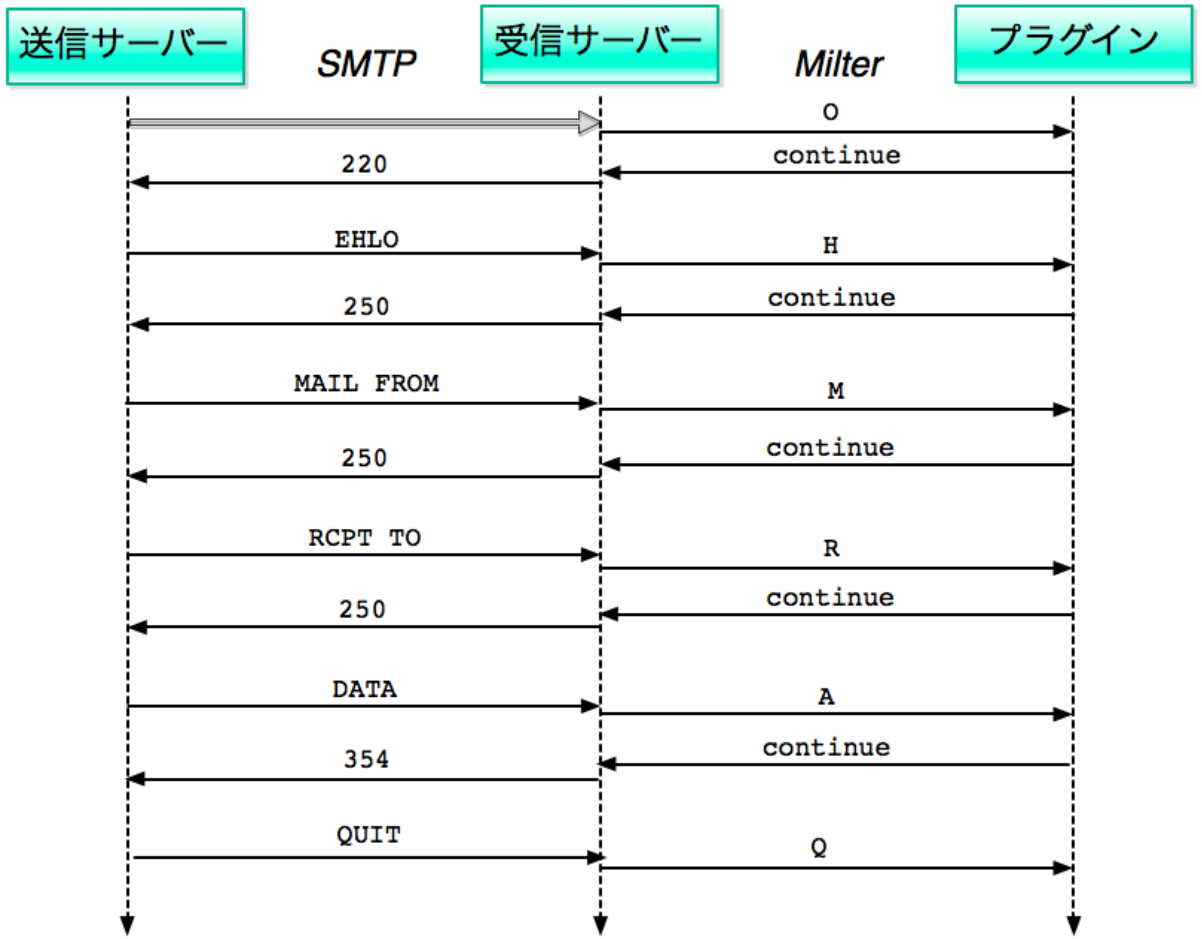
ポリシー
記述言語

サイトごとのポリシーを実現

Haskell

パーサーの実装が容易
型安全なプログラミング
遅延評価

対話的な Milter プロトコル

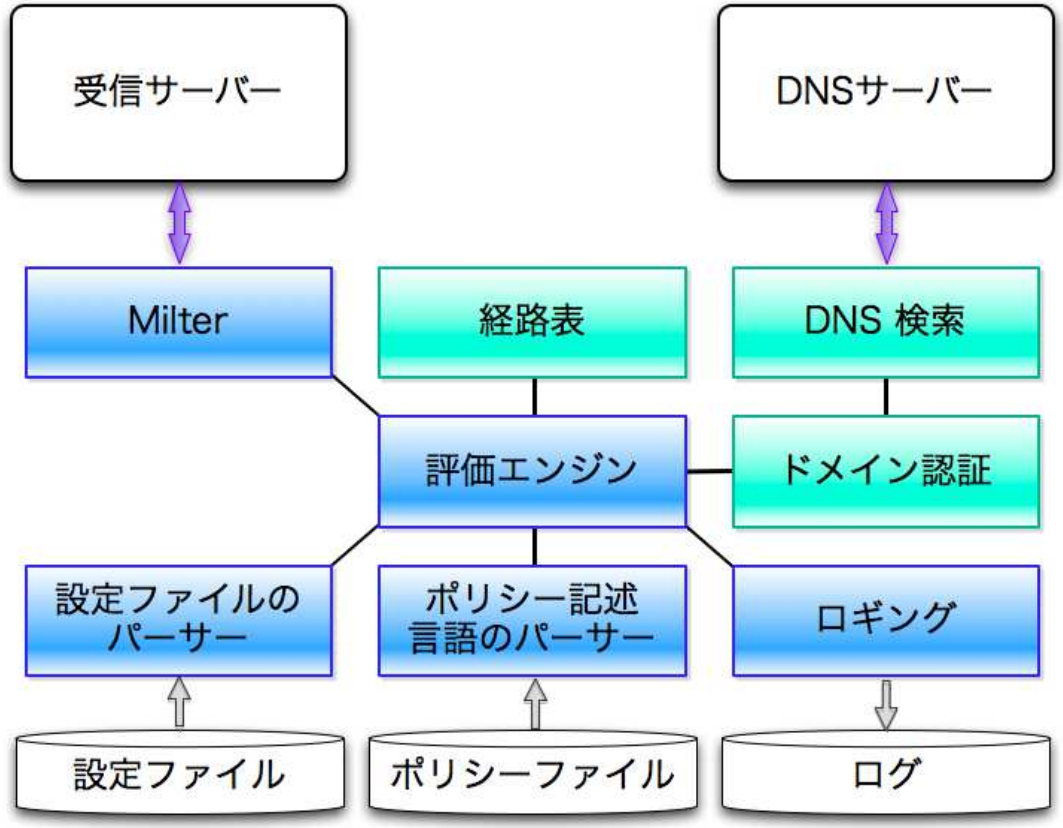


ポリシー記述言語

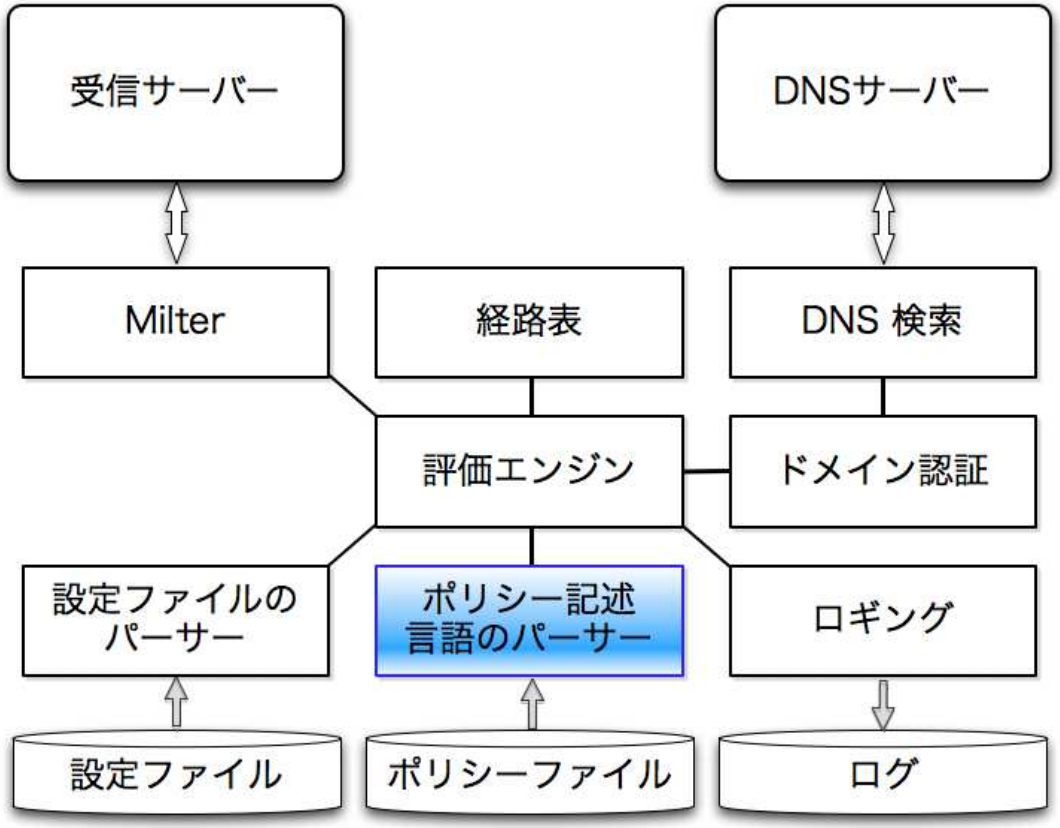
```
connect {
    accept: #ip == 127.0.0.1;
    continue;
}
mail_from {
    accept: #spf == pass;
    continue;
}
header {
    accept: #sender_id == pass;
    reject: #mail_from == "yahoo.com"
         && #sig_domainkeys == No;
    continue;
}
body {
    accept: #dkim == pass;
    accept: #domainkeys == pass;
    continue;
}
```

実装の特徴的な点

実装の全体像


受信サーバー
DNSサーバー
Milter
経路表
DNS 検索
評価エンジン
ドメイン認証
設定ファイルの
パーサー
ポリシー記述
言語のパーサー
ロギング
設定ファイル
ポリシーファイル
ログ

ポリシー記述言語のパーサー



ポリシー記述言語のパーサー

- ポリシーの例

```
#ip == 127.0.0.1, ::1
```

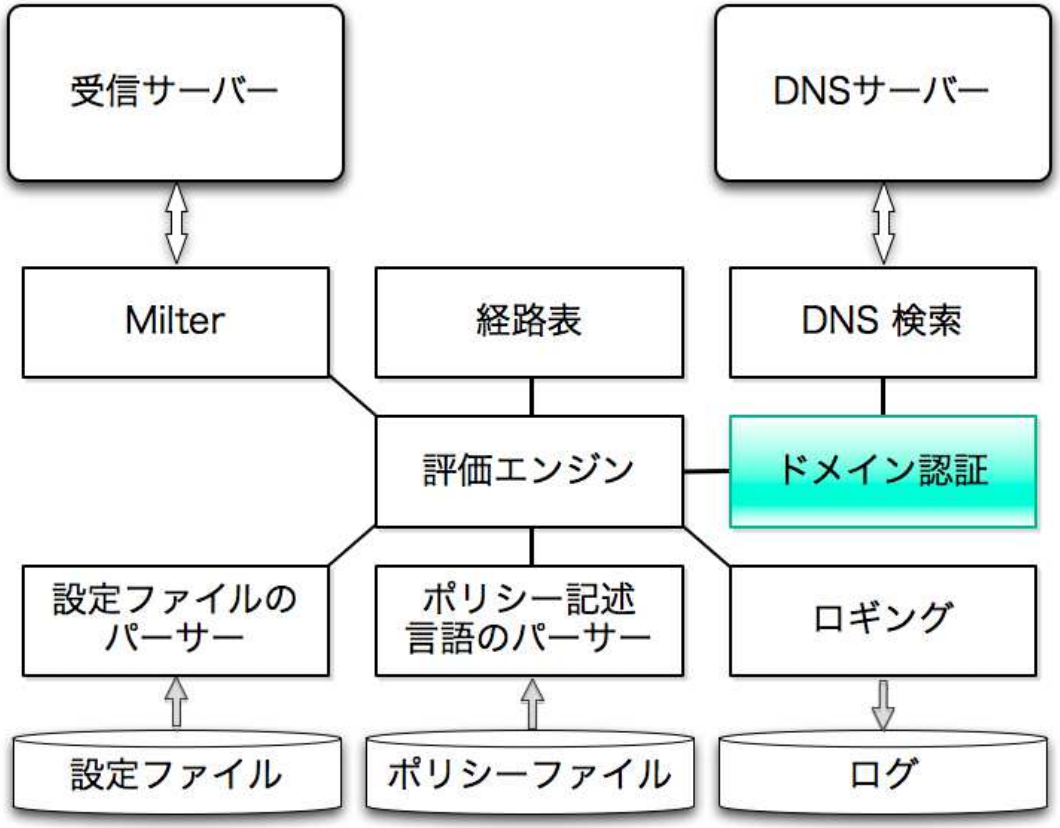
- Haskell での実装

```
term = do
    char '#'
    var@(vtyp,vid) <- variable
    opr <- opMatch <|> opNotMatch
    cst@(ctyp,cval) <- constant
    return $ var `opr` cst

constant = ipList <|> domainList <|> resultList2
       <|> definedConstant <|> yesOrNo
```

- 強力な型検査 (後述)

ドメイン認証ライブラリー



IP方式の危険な設定

ループ

```
example.jp = +ip4:192.0.2.1
             redirect=example.jp
```

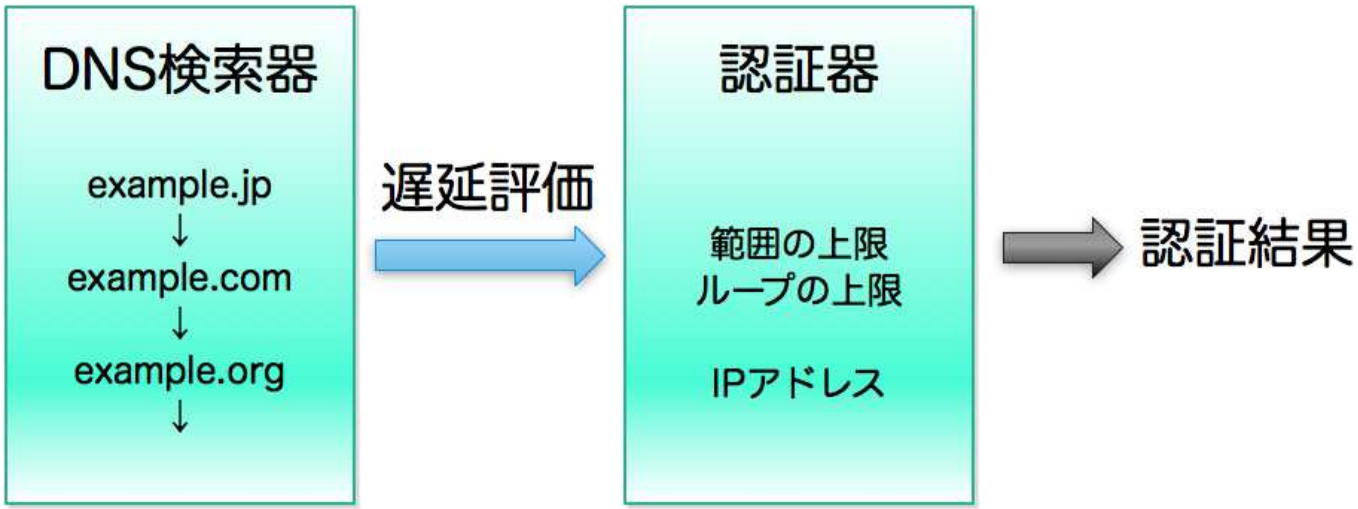
大きすぎる範囲

```
example.jp = +all
example.jp = +0.0.0.0/0
```

- 一度にたくさんの仕事をするとコードが複雑になる
  - DNS の検索、ドメインの認証
  - ループ判定、大きすぎる範囲の排除

遅延評価によるコードの分離

- DNS 検索器
  - ループの可能性を気にせず DNS を検索する



- 遅延評価
  - 必要にならない値は生成されない
  - 不要なドメインは検索しない

4種類のテスト

テスト

コンパイルによる
型検査

ユニットテスト

性質テスト
テストケースの自動生成

フィールドテスト

コンパイルによる型テスト

- Haskell の文法は制約が強い

```
collatz :: Int -> Int
collatz n = if even n
            then n ‘div‘ 2
            -- 必ず else 節が必要
            else n * 3 + 1
```

- 再帰の末端では型検査を受ける

```
lines :: String -> [String]
lines "" = []
lines s = case s’ of
    []      -> [l]
    (_:s’’) -> l : lines s’’
  where
    (l, s’) = break (== ’\n’) s
```

- コンパイルがテスト駆動開発を支援する

ユニットテスト

- ドメイン認証ライブラリ
  - 存在しないドメインではエラーとなるかのテスト

```
test_ipv4_2 :: Assertion
test_ipv4_2 = do
    rs <- makeResolvSeed defaultResolvConf
    withResolver rs $ \resolver ->
      runSPF defaultLimit resolver
        "exapmle.org" ip >>= (@?= DATempError)
  where
    ip = IPv4 . read $ "192.0.2.1"
```

性質テスト

- 関数の性質を記述する
  - テストケースは QuickCheck が自動生成する
- 経路表
  - 値を挿入したら、その値は検索できなければならない

```
ins_look :: AddrRange a -> [AddrRange a] -> Bool
ins_look k xs = lookup k (insert k k t) == Just k
  where
    t = fromList rs $ zip xs xs
```

フィールドテスト

- Mew.org で作動中
  - Mew.org へメールを転送しているユーザーは存在しない
- ログ解析
  - 2010年7月21日～27日
  - 486,405 個のTCPコネクション中、10,332 個が正常な SMTP

ルール	メールの数	比率
`#spf == pass`	1,015	9.8 %
`#sender_id == pass`	47	0.5 %
`#mail_from == "yahoo.com" && #sig_domainkeys == No`	144	1.4 %
`#dkim == pass`	0	0 %
`#domainkeys == pass`	0	0 %
ルールに合致しない	9,126	88.3 %
合計	10,332	100.0 %

評価

- 目的はドメイン認証を受信側へ普及させること
- 普及率はアンケートなどでしか調査できない
- 評価は難しい

ソフトウェアの公開

- Receiver Policy Framework のマニュアルなど
  - http://www.mew.org/~kazu/proj/rpf/
- ソース
  - github
- パッケージの配布元
  - hackageDB
- インストール
  - Haskell Platform
  - `% cabal install rpf`