ANCHORING TRUST

REWRITING DNS FOR THE SEMANTIC NETWORK WITH RUBY AND RAILS

http://slides.games-with-brains.net/

INTRODUCTION

WE ARE SCIENTISTS

ROMEK SZCZESNIAK
CRYPTOGRAPHER
ROMEK@SPIKYBLACKCAT.CO.UK

ELEANOR MCHUGH
PHYSICIST
ELEANOR@GAMES-WITH-BRAINS.COM

WE DO SCIENCE!

- applied research
  - semantic DNS infrastructure
  - identity & access management
- *Rails*-abuse, *Ruby*-fu, vapourware
  - the *wraith* network packet sniffer
  - the *rindr* DNS application server

THE WEASEL WORDS

- danger! *experimental* code and concepts ahead

- all such code is provided for entertainment purposes only and should be used with extreme caution, under adult supervision, blah blah blah...

- any resemblance to *actual code & concepts™*, living or dead, is purely coincidental

THE SEMANTIC NETWORK

TODAY'S INTERNET

- packet switched network
- based on Internet Protocol
- each node has an IANA-allocated IP address
- some IP addresses map to domain names
- all domains ultimately map to IP addresses
- nominally application agnostic

THE DNS SYSTEM

- defined in RFC 1034-1035 (November 1987)
- resolves IP addresses from domain names
- predominantly a static system
- ISC BIND 9 is the base implementation
- alternatives: NSD; MaraDNS; djbdns

A DNS LOOKUP

- the following is the output of querying the domain spikyblackcat.co.uk using **dig**

```
; <<>> DiG 9.3.4 <<>> spikyblackcat.co.uk
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 30042
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;spikyblackcat.co.uk.          IN      A

;; ANSWER SECTION:
spikyblackcat.co.uk.    3600    IN      A       89.202.129.47

;; Query time: 276 msec
;; SERVER: 208.67.222.222#53(208.67.222.222)
;; WHEN: Mon Aug 20 19:38:13 2007
;; MSG SIZE  rcvd: 53
```

SEMANTIC NETWORKING

- based on names rather than addresses
- exploits the dynamism in modern DNS
- not all domain names resolve to IP addresses
- embodies rich associations and content
- application and service oriented
- some parallels with semantic networks in AI

ENUM

TELEPHONE NUMBER MAPPING

- bridges the Internet and PSTN via DNS
- service-oriented
- RFCs 3401-3405 and RFC 3761
- Conroy and Fujiwara's ENUM experiences
- **D**ynamic **D**elegation **D**iscovery **S**ystem
- **N**aming **A**uthority **P**oin**T**e**R**s

EXAMPLE ENUM DOMAIN

01794833666 → 6.6.6.3.3.8.4.9.7.1.4.4.e164.arpa

6.6.6.3.3.8.4.9.7.1	is the phone number to connect to
.4.4	is the country dialing code for the UK
.e164	identifies the telephone numbering system
.arpa	is the internet infrastructure equivalent of .com

5	IN NAPTR	10	6	"U"	"E2U+web:http"	"!^.*$!http:maps.google.com/maps?output=png&q=SO51%2BUK&btnG=Search!"	.
5	IN NAPTR	10	6	"U"	"E2U+web:sip"	"!^.*$!sip:3666@roke.co.uk!"	.
5	IN NAPTR	10	6	"U"	"E2U+x-skype:callto"	"!^.*$!callto:lconroy!"	.
5	IN NAPTR	10	6	"U"	"E2U+voice:tel"	"!^.*$!tel:+441794833666!"	.
5	IN NAPTR	10	6	"U"	"E2U+voice:tel"	"!^.*$!tel:+441794364902!"	.
5	IN NAPTR	10	6	"U"	"E2U+voice:sip"	"!^.*$!sip:3671@insensate.co.uk!"	.

THE NAPTR RESOURCE

NAMING AUTHORITY POINTER

- specified in RFCs 2915 and 3403
- allows URI rewriting using regular expressions
- two modes of operation
  - a terminal record replaces one URI with another when a regular expression is matched
  - a non-terminal record redirects the DNS resolution to a specified domain name

THE NAPTR RESOURCE

NAMING AUTHORITY POINTER

- this table shows the two modes of NAPTR as they appear in a DNS zone record

TTL			order	preference	flag	service type	regex + replacement	terminator
600	IN	NAPTR	100	50	“u”	“E2U+sip”	“!^.*$!sip:joe@fish.com!”	.
600	IN	NAPTR	100	51	“”	“”	“”	test.com

TTL	time in seconds before record must be resolved from an authoritative server
order	order in which records should be evaluated
preference	preference within a given order index
flag	“u” for a standard terminal record resource record as specified in the RFCs
service type	ENUM to URI + service type for an ENUM-specific service type
regex	regular expression to use for matching
replacement	string to replace the matched URI with
terminator	either “.” or the target domain name for a non-terminal record

RUBY NAPTR SUPPORT

- Telnic library
  - developed in 2006
  - open-source - available on enquiry
- *rindr* library
  - many lessons learnt from Telnic library
  - supports enhanced NAPTR functionality

SO WHAT IS RINDR?

- a Ruby DNS server
  - UDP/TCP network server
  - X(HT)ML and DNS protocols
  - application-oriented NAPTR extensions
- Rails integration via BackgrounDRb
- will be published under an MIT license

RINDR DNS EXTENSIONS

- DNS as a generic publishing infrastructure
- adds a relational data model to DNS
- backwards compatible for interoperability
- **I**dentity and **A**ccess **M**anagement system
- **D**omain **M**arkup **L**anguage

RAILS & THE NETWORK

- Rails uses HTTP(S)
- Ruby has good support for other protocols
- Rails is a single-threaded framework
- network access blocks current thread
- how about using a task server?

BACKGROUNDRB

- task-server based on DRb
- multi-threaded so non-blocking
- MiddleMan objects acts as task proxy
- Rails application polls task proxy

NETWORKING PRIMER

- feature-complete Ruby examples:
  - UDP
  - TCP
    - TCPServer
    - GServer
- use BackgrouDRb for Rails integration

SERVING UDP

```
require 'socket'
require 'thread'
require 'mutex_m'

class UDPServer
 include Mutex_m
 attr_reader   :address, :port, :log, :timer

 def initialize address, port
  @address, @port = address, port
  @workers = []
  @timer = 0
 end

 def start
  @socket = UDPSocket.new
  @socket.bind(@address, @port)
  @socket.setsockopt(Socket::SOL_SOCKET, Socket::SO_REUSEADDR, 1)
  spawn_watchdog
  event_loop :logging => true
 end

 def stop
  @workers.each { |thread|  thread.kill }
  lock
   @socket.close
   @socket = nil
  unlock
  puts "server stopped"
 end

 def active?
  @socket ? true : false
 end

 def serve request
  ["hello", 0]
 end

 def watchdog
 end

private
 def spawn_watchdog
  Thread.new {
   STDOUT.puts "watchdog thread spawned"
   loop do
    sleep 1
    @timer += 1
    watchdog if @socket
   end
   }
 end

 def event_loop options = {}
  @logging_enable = options[:logging]
  puts "server started"

  loop do
   if sockets = select([@socket]) then
    sockets[0].each do |s|
     @workers << Thread.new(s) do |socket|
      message, peer = *socket.recvfrom(512)
      report(message)
      reply, status = *serve(message)
      report(reply)
      UDPSocket.open.send(reply, status, peer[2], peer[1])
     end
    end
    @workers.compact!
   end
  end
 end
end
```

BEING A UDP CLIENT

generic client

```
require 'socket'

EndPoint = Struct.new(:host, :port)

class UDPClient
  attr_reader   :remote, :status

  def initialize local_host, local_port
    @local = EndPoint.new(local_host, local_port)
  end

  def connect remote_host, remote_port
    raise if @socket
    puts "starting client"
    @remote = EndPoint.new(remote_host, remote_port)
    @socket = UDPSocket.open
    @socket.bind(@local.host, @local.port)
    @socket.send("", 0, @remote.host, @remote.port)
  end

  def send message
    @socket.send(message, 0, @remote.host, @remote.port)
  end

  def receive max_bytes = 512
    raise unless @socket
    message, @status = *(@socket.recvfrom(max_bytes))
    message
  end

  def disconnect
    @socket.close
    @socket = nil
  end
end
```

sample server and client

```
class ReverseServer < UDPServer
  attr_reader :message_count

  def initialize address, port
    super
    @message_count = 0
  end

  def serve request
    lock
      @message_count += 1
    unlock
    [request.reverse, 0]
  end
end

server = ReverseServer.new("localhost", 3000)
Thread.new do ||
  server.start
end

while server.active?
  while server.log.length > 0
    puts server.log.shift
  end
  sleep 1
end
puts "server halted"
```

```
class TimeClient < UDPClient
  def send
    message = "#{Time.now}"
    puts "sending message: #{message}"
    super message
    puts "received response #{receive()}"
  end
end


client = TimeClient.new("localhost", 3001)
client.connect("localhost", 3000)
(1..300).each do |i|
  client.send
end
client.disconnect
```

```
starting client
sending message: Mon Sep 03 00:42:31 +0100 2007
received response 7002 0010+ 13:24:00 30 peS noM
sending message: Mon Sep 03 00:42:31 +0100 2007
received response 7002 0010+ 13:24:00 30 peS noM
sending message: Mon Sep 03 00:42:31 +0100 2007
received response 7002 0010+ 13:24:00 30 peS noM
sending message: Mon Sep 03 00:42:31 +0100 2007
received response 7002 0010+ 13:24:00 30 peS noM
sending message: Mon Sep 03 00:42:31 +0100 2007
received response 7002 0010+ 13:24:00 30 peS noM
```

GSERVER V. TCPSERVER

- GServer is convenient to use
  - subclass and overload the **serve** method
  - managed thread pool for easy multi-tasking
- TCPServer is flexible, allowing for more complex server designs

USING TCPSERVER

```
require 'socket'
require 'thread'
require 'mutex_m'

class EnchancedTCPServer
 include Mutex_m
 attr_reader   :address, :port, :log, :timer

 def initialize address, port
  @address, @port = address, port
  @log = []
  @workers = []
  @timer = 0
  @logging_enabled = false
 end

 def start
  @server = TCPServer.new(@address, @port)
  spawn_watchdog
  event_loop :logging => true
 end

 def stop
  @workers.each { |thread|  thread.kill }
  lock
   @server = nil
  unlock
  puts "server stopped"
 end

 def active?
  @server ? true : false
 end

 def on_connect session; end
 def on_disconnect session; end
 def on_serve session;  end
 def on_watchdog;  end

 private
  def spawn_watchdog
   Thread.new {
    loop do
     sleep 1
     @timer += 1
     on_watchdog if @server
    end
    }
  end

  def event_loop options = {}
   @logging_enable = options[:logging]
   loop do
    on_connect(session = @server.accept)
    @workers << Thread.new(session) do |session|
     loop do
      if session.eof? then
       on_disconnect(session)
       break
      else
       on_serve(session)
      end
     end
     session.close
    end
    @workers.compact!
   end
  end

  def report message
   if @logging_enabled then
    lock
     @log << message.dup
    unlock
   end
  end
end
```

BEING A TCP CLIENT

generic client

```
require 'socket'

EndPoint = Struct.new(:host, :port)

class TCPClient
 attr_reader   :remote, :status

 def connect remote_host, remote_port
  raise if @socket
  puts "starting client"
  @remote = EndPoint.new(remote_host, remote_port)
  @socket = TCPSocket.new(@remote.host, @remote.port)
 end

 def send message
  @socket.puts(message)
  on_response
 end

 def on_response;  end

 def receive
  raise unless @socket
  begin
   response = @socket.gets
  end until response
  response
 end

 def disconnect
  @socket.close
  @socket = nil
 end
end
```

sample server and client

```
class ReverseServer < EnchancedTCPServer
 attr_reader :message_count, :max_count

 def initialize address, port
  super
  @message_count = 0
 end

 def on_serve session
  @message_count += 1
  message = (session.gets)[0..-2]
  session.puts "reversing #{message}"
  session.puts message.reverse
 end
end

server = ReverseServer.new("localhost", 3000)
Thread.new do ||
 server.start
end

while server.active?
 while server.log.length > 0
  puts server.log.shift
 end
 sleep 1
#  puts "Timer count: #{server.timer} / #
{server.max_count}"
end
puts "server halted"
```

```
class TimeClient < TCPClient
 def send
  message = "#{Time.now}"
  puts "sending message: #{message}"
  super message
 end

 def on_response
  line = receive
  puts "received response: #{line}"
 end
end


client = TimeClient.new
client.connect("localhost", 3000)
(1..2).each do |i|
 client.send
 client.on_response
 puts "=========="
 sleep 1
#  client.receive
#  sleep 1
end
client.disconnect
```

```
starting client
sending message: Tue Sep 04 19:27:11 +0100 2007
received response: reversing Tue Sep 04 19:27:11 +0100 2007
received response: 7002 0010+ 11:72:91 40 peS euT
==========
sending message: Tue Sep 04 19:27:12 +0100 2007
received response: reversing Tue Sep 04 19:27:12 +0100 2007
received response: 7002 0010+ 21:72:91 40 peS euT
==========
```

THE GSERVER WAY

generic client

```
require 'socket'

EndPoint = Struct.new(:host, :port)

class TCPClient
 attr_reader   :remote, :status

 def connect remote_host, remote_port
  raise if @socket
  puts "starting client"
  @remote = EndPoint.new(remote_host, remote_port)
  @socket = TCPSocket.new(@remote.host, @remote.port)
 end

 def send message
  @socket.puts(message)
  on_response
 end

 def on_response;  end

 def receive
  raise unless @socket
  begin
   response = @socket.gets
  end until response
  response
 end

 def disconnect
  @socket.close
  @socket = nil
 end
end
```

sample GServer and client

```
require 'gserver'

class ReverseServer < GServer
 def initialize host, port
  super port, host
 end

 def serve client
  puts "serving client: #{client.peeraddr[2]}"
  request = (client.gets)[0..-2]
  puts request
  client.puts("request => #{request}")
  client.puts("response => #{request.reverse}")
 end
end

server = ReverseServer.new("localhost", 3000)
server.start
server.join
```

```
class TimeClient < TCPClient
 def send
  message = "#{Time.now}"
  puts "sending message: #{message}"
  super message
 end

 def on_response
  line = receive
  print "#{line.join("")}"
 end
end

client = TimeClient.new
(1..2).each do |i|
 client.connect("localhost", 3000)
 client.send
 client.on_response
 puts "=========="
 client.disconnect
 sleep 1
end
```

```
starting client
sending message: Tue Sep 04 22:17:27 +0100 2007
request => Tue Sep 04 22:17:27 +0100 2007
response => 7002 0010+ 72:71:22 40 peS euT
==========
starting client
sending message: Tue Sep 04 22:17:29 +0100 2007
request => Tue Sep 04 22:17:29 +0100 2007
response => 7002 0010+ 92:71:22 40 peS euT
==========
```

ANCHORING TRUST IN DNS

LDAP

LIGHTWIGHT DIRECTORY ACCESS PROTOCOL

- International Telecommunications Union
- X.500 directory services over TCP/IP
- based on a tree of directory entries
- each directory entry is a set of attributes
- often used as an authentication repository
- Ruby/LDAP

SINGLE SIGN-ON

- distributed authentication infrastructure
  - identity providers authenticate
  - service providers serve client requests
- various authentication schemes
  - OpenID accepts a user-specified URI
  - Kerberos issues a ticket on successful login

BENEFIT OF USING DNS

- globally accessible
- DNS as the anchor for online identity
- individual and group **A**ccess **C**ontrol **L**ists
- prioritised contact mechanisms
- supports software-oriented routing

DNS INSECURITIES

- publicly viewable
- globally accessible
- lacks client authentication
- server authentication via DNSSEC unwieldy

THE .TEL SOLUTION

- ICANN authorised Top-Level Domain
- uses domain names, i.e. romek.tel
- globally accessible single point of contact
- contact details stored as NAPTR records
- provides a *friending* service for access control
- proprietary DNS privacy model

EXAMPLE .TEL ENTRY

- romek.tel is provisioned with the following entries, with the .tel default TTL of 60 seconds

100	100	“u”	“E2U+sip”	“!^.*$!sip:3672@insensate.co.uk!”	.
100	103	“u”	“E2U+email:mailto”	“!^.*$!mailto:romek@telnic.org!”	.
100	110	“u”	“E2U+sms:tel”	“!^.*$!tel:+447833673805!”	.
100	112	“u”	“E2U+web:http”	“!^.*$!http://www.telnic.org!”	.
100	112	“u”	“E2U+web:https”	“!^.*$!https://www.telnic.org!”	.
100	113	“u”	“E2U+ft:ftp”	“!^.*$!ftp://ftp.telnic.org!”	.
100	114	“u”	“E2U+fax:tel”	“!^.*$!tel:+447833673805!”	.
100	115	“u”	“E2U+sms:tel”	“!^.*$!tel:+447833673805!”	.
100	116	“u”	“E2U+ems:tel”	“!^.*$!tel:+447833673805!”	.
100	117	“u”	“E2U+mms:tel”	“!^.*$!tel:+447833673805!”	.
100	118	“u”	“E2U+voice:tel”	“!^.*$!tel:+447833673805!”	.
32767	32500	“”	“”		0.0.0.0.8.2.8.7.0.2.1.4.4.e164.arpa.
32767	32501	“”	“”		lookup.telnic.org.

DNS AS DATA STORE

- storage of application-specific data
- pioneered by Project Athena's Hesiod
  - unordered TXT **R**esource **R**ecords
  - publication of UNIX /etc/passwd file
- NAPTR RRs ordered and structured
- DNS as persistent relational data store

GENERIC PRIVATE DATA

- non-proprietary private data in DNS
- private data with access control
- each user has a public/private key-pair
- each record is encrypted with a session key
- key-pairs are used to transmit the session key
- no session key, no access!!!

NAPTR CRYPTO RECIPE

- take a standard NAPTR record

IN	NAPTR	100	11	“u”	“E2U+email:mailto”	“!^.*$!mailto:romeks@gmail.com!”	.

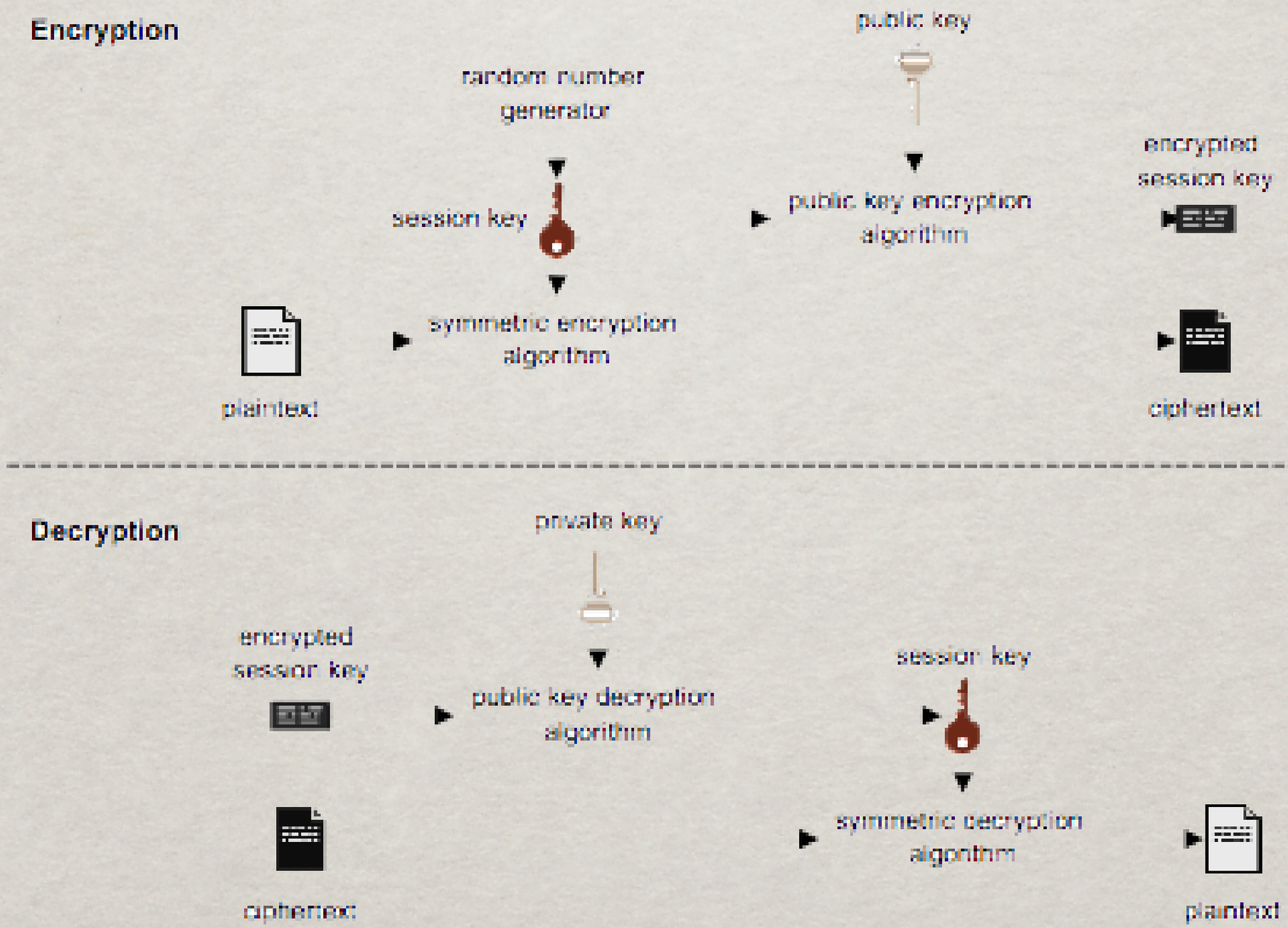
- encrypt with session key & initialisation vector

initialisation vector	+FgTpo7SPyd7cZx+cGVAtg==
session key	/wVEQmHS4vhwO/AJTDqpGoXwMYYHiSUmZShY7GSCcrl=

- store results in an encrypted replacement field

IN	NAPTR	100	11	“u”	“E2U+email:mailto”	“!^.*$! HVnGeCBG4ISOvghq8jwylpFQmvotfaSjdgQ88ExkaIU=!”	.

A QUICK DIAGRAM



THE DOWNSIDE OF DNS

- globally accessible
  - distributed key cracking
  - private data cached anywhere
- IETF standards evolve slowly
- RR required for encrypted session key
- trade-off between caching and update speed

RUBY & CRYPTOGRAPHY

- supports the OpenSSL library
- Ruby Kaigi 2006 demonstration
- but still not ready for prime-time
  - doesn't address problems in OpenSSL
  - no support for HSMs
  - no pure Ruby crypto library

PUBLIC KEY CRYPTO

RIVEST SHAMIR ADLEMAN

```
require 'openssl'
require 'base64'

class RSAKey
 PATTERNS = { :key => /^key: /   }

 def initialize
  @key = nil
 end

 def create bits = 256
  @key = OpenSSL::PKey::RSA.new(bits) { print "." }
  puts
 end

 def public_key
  @key.public_key.to_pem
 end

 def private_key
  @key.to_pem
 end

 def key= pem_data
  @key = OpenSSL::PKey::RSA.new(pem_data)
 end

 def encrypt text
  @key.public_encrypt(text)
 end

 def decrypt text
  @key.private_decrypt(text)
 end
end
```

```
key = RSAKey.new
key.create
puts "public key:     #{key.public_key}"
puts "private key:    #{key.private_key}"
puts "plain text:     #{text = 'something very secret'}"
puts "encrypted text: #{encrypted_text = Base64.encode64(key.encrypt(text))}"
puts "decrypted text: #{key.decrypt(Base64.decode64(encrypted_text))}"
```

```
public key:     -----BEGIN RSA PUBLIC KEY-----
                MCgCIQC6ovYcthdixInQTI9jQom5JkoYGHm
                +kS6H8iXw01TZyQIDAQAB
                -----END RSA PUBLIC KEY-----
private key:    -----BEGIN RSA PRIVATE KEY-----
                MIGrAgEAAiEAuqL2HLYXYsSJ0EyPY0KJuSZKGBh5vp
                Euh/Il8NNU2ckCAwEAAQIh
                ALhg1ZJ3sZK5ZwyJFf6RdUvrvcV1JfVpJJW/g/
                FT1YQBAhEA5gFML0nxGBgDr7ya
                AU+b8QIRAM+646/
                S648QgsnCYXMg01kCEEzSCCEPQEA83RZYFtPzQ/
                ECEBTUvPho
                wk56bWMmJveQlyECEQC6zIXUHHH04PwmQHERy08c
                -----END RSA PRIVATE KEY-----
plain text:     something very secret
encrypted text: Z6mlEfMOVgyst3z9/ps9bF0ZD7myW9WPtjAKCAd7GFw=
decrypted text: something very secret
```

SYMMETRIC CRYPTO

ADVANCED ENCRYPTION STANDARD

```
require 'openssl'
require 'base64'
require 'digest/md5'
require 'digest/sha1'
require 'digest/sha2'

class AESDataStore
 attr_reader :key, :encrypted_text, :initialisation_vector

 def initialize key = nil, vector = nil
  @cipher = OpenSSL::Cipher::Cipher.new("aes-256-cbc")
  @encrypted_text = ""
  create_key(key, vector)
 end

 def create_key key = nil, vector = nil
  @key = (key || OpenSSL::Random.random_bytes(256/8))
  @initialisation_vector = (vector || @cipher.random_iv)
end

def encrypt data
 @cipher.encrypt
 @encrypted_text = run_cipher(data)
end

def decrypt
 @cipher.decrypt
 run_cipher(@encrypted_text)
end

def export_key file_name = "aes-256-cbc.key"
 key_file = File.new(file_name, "w")
 key_file << "key: #{@key}\n"
end

private
 def run_cipher data
  @cipher.key = @key
  @cipher.iv = @initialisation_vector
  cipher_text = @cipher.update(data)
  (cipher_text << @cipher.final)
 end
end

text = "sample plain-text"
encrypted_data_store = AESDataStore.new
encrypted_data_store.instance_eval do
 create_key
 export_key
 puts "original message:  #{text}"
 puts "encrypted message: #{Base64.encode64(encrypt(text))}"
 puts "decrypted message: #{decrypt()}"
 puts "key:              #{@key}"
 puts "vector:           #{@initialisation_vector}"
end
```

```
original message:       sample plain-text
encrypted message:      qbPa75G+84M4+zc0ayiX4tttbtHPhhvooHRptMxDl6M=
decrypted message:      sample plain-text
key:                    ???  {??e??T??XM??6?Q??*@?z?
vector:                 ???  ??<I????P
```

A NETWORK KEY SERVER

a simple server

```
require 'shared'
require 'gserver'

class AESServer < GServer
 attr_reader   :aes_keys, :acl, :server_key

 def initialize host, port
  super port, host
  @server_key = RSAKey.new
  @aes_keys = {}
  @acl = {}
end

def create_key key_name
 @aes_keys[key_name] = AESDataStore.new
end

def authorise host, key_name
 (@acl[key_name] ||= []) << host
end

def serve client
 case client.get_line
 when "public"
  client.puts @server_key.public_key.encode

 when "aes"
  authorise_action(client, @acl) do |client, key_name|
   raise unless @aes_keys[key_name]
   client_key = RSAKey.new(client.get_line.decode)
   aes_key = @aes_keys[key_name].key
   client.puts client_key.encrypt(aes_key).encode
  end
 end
end

private
 def authorise_action client, acl
  host = client.peeraddr[3]
  key_name = client.get_line
  if acl[key_name].include?(host) then
   yield client, key_name if block_given?
  end
 end
end

server = AESServer.new("localhost", 3000)
server.instance_eval do
 create_key "1"
 create_key "2"
 create_key "3"
 authorise "::1", "1"
 authorise "::1", "2"
 start
 join
end
```

A NETWORK KEY SERVER

convenience functions

```
require 'socket'
require 'openssl'
require 'base64'

class IO
  def get_line
    (line = gets) ? line.chop : line
  end
end

class AESDataStore
  attr_reader :key, :encrypted_data

  def initialize key = nil
    @cipher = OpenSSL::Cipher::Cipher.new("aes-256-cbc")
    @encrypted_data = ""
    @key = (key || OpenSSL::Random.random_bytes(256/8))
  end

  def encrypt data
    @cipher.encrypt
    @encrypted_data = run_cipher(data)
  end

  def decrypt
    @cipher.decrypt
    run_cipher(@encrypted_data)
  end

private
  def run_cipher data
    @cipher.key = @cipher.iv = @key
    cipher_text = @cipher.update(data) + @cipher.final
  end
end
```

```
class RSAKey
  def initialize pem_data = nil
    @key = if pem_data then
      OpenSSL::PKey::RSA.new(pem_data)
    else
      OpenSSL::PKey::RSA.new(768))
    end
  end

  def public_key
    @key.public_key.to_pem
  end

  def private_key
    @key.to_pem
  end

  def encrypt plain_data
    @key.public_encrypt(plain_data) rescue ""
  end

  def decrypt plain_data
    @key.private_decrypt(plain_data)
  end
end

class String
  def decode
    Base64.decode64(self)
  end

  def encode
    Base64.encode64(self).tr("=\n", "")
  end
end
```

THE RAILS MIDDLEMAN

the client rewritten with BackgrounDRb

```
class KeyServerWorker < BackgrounDRb::Worker::Base
  attr_reader   :client_key

  def do_work args
    @client_key ||= RSAKey.new
    logger.info('requesting keys from server')
    results[:request_time] = Time.now
    results[:completed] = false
    load_server_key
    args.each do |key_name|
      results[key_name] = get_aes_key(key_name)
    end
    results[:duration] = Time.now - results[:request_time]
    results[:completed] = true
  end

  def load_server_key
    @server_key = nil
    connect do |server|
      send "public"
      results[:server_key] = RSAKey.new(@socket.get_line.decode))
    end
  end

  def get_aes_key key_name
    connect do |server|
      send_with_key "aes", name
      results[:socket].get_line.decode rescue nil
    end
  end

  private
    def connect
      raise if results[:socket]
      begin
        results[:socket] = TCPSocket.new("localhost", 3000)
        value = yield(results[:socket]) if block_given?
      ensure
        results[:socket].close
        results[:socket] = nil
      end
      value
    end

    def send message, *params
      results[:socket].puts(message, *params)
    end

    def send_with_key message, *params
      send message, *(params << @client_key.public_key.encode)
    end
  end
  KeyServerWorker.register
```

BRIDGING RAILS

accessing the BackgrounDRb client from Rails

```
class KeyServerController < ApplicationController
  def start_background_task
    session[:job_key] = MiddleMan.new_worker(:class => :key_server_worker, :args => (1..3).to_a)
  end

  def keys
    if request.xhr?
      render :update do |page|
        page.call('list_keys', results["1"], results["2"], results["3"])
        MiddleMan.delete_worker(session[:job_key]) if results[:completed]
      end
    else
      redirect_to :action => 'index'
    end
  end
end
```

CONCLUSION

A NEW INTERNET

- the Internet is evolving
  - dynamic, semantic name resolution
    - Apache-style URI rewriting in DNS
    - black-box data storage
- AJAX-HTTP-DNS integration
- attend our BoF in **Salon 3** (19:30-20:30)