Experimenting with Distributed Generation of RSA Keys

Thierry Congos, François Lesueur
firstname.lastname@supelec.fr

SUPÉLEC, SSIR Group (EA 4039), France

STM, September 24 2009
Saint-Malo, France
Security in MANET/P2P Networks

Specificities of MANET/P2P Networks

*Dynamic and Collaborative networks without Central Authority*

**Approach**

1. Cooperative admission control to the network
2. Security protocols tolerating a bounded number of attackers
Certification to Enforce Security Properties

Traditional View
- Security is enforced by a central point
- Capacities are proved by certificates
⇒ Certification Authorities, centralization

Our Context: Distributed Certification
- Capacities are still proved by certificates
- These certificates are signed collaboratively by members
⇒ Threshold Cryptography, no center
Usages of Distributed Certification

### Availability of the CA
- Once initialized, no more central point
- Certification available if:
  - Partition of the network
  - Loss of connectivity

### No central point of trust
- Certificates materialize agreement of some peers
- No single entity can forge certificates

⇒ Key must be distributedly generated!
Outline

1. Background
2. Evaluation on Current Hardware
3. Conclusion
Background
RSA Key Generation

- Generate $p$ and $q$ $l$-bit primes
- Compute $N = p \times q$
- Compute the totient $\varphi(N)$
- Choose $e$ such as $1 < e < \varphi(N)$ and $e$ coprime with $\varphi(N)$
- Determine $d$ such as $d \times e \equiv 1 \pmod{\varphi(N)}$

Public key is $(e, N)$
Private key is $(d, N)$
Distributed RSA Key Generation

- $k$ parties generate the key
- At the end of the computation, each party $i$ knows:
  - the modulus $N$
  - the public exponent $e$
  - a private share $d_i$

- $d = \sum_{i=1}^{k} d_i$
- $d$, $p$ and $q$ are not known by anyone
# Distributed RSA Key Generation Algorithm (Boneh and Franklin) 1/2

## 1. Generate $p$ and $q$

- Each party generates $p_i$ and $q_i$
  
- $p = \sum_{i=1}^{k} p_i$ and $q = \sum_{i=1}^{k} q_i$

- $p$ and $q$ are not explicitly computed

## 2. Compute $N$

- BGW protocol computes $N = p \times q$ from $p_i$ and $q_i$

- $p$ and $q$ are not revealed
3. Test $N$ for bi-primality
   - $N$ is tested for bi-primality by each party
   - If $N$ is not a product of two primes, start again…

4. Generate shares
   - Each party obtains a share $d_i$
   - $d = \sum_{i=1}^{k} d_i$
Previous Evaluations

Malkin, Wu and Boneh [SNDSS 99]
- 333Mhz Pentium II
- LAN/WAN
- 5 entities on LAN, 3 on WAN

Wright and Spalding [SODA 99]
- 3 servers on the same machine
- Impacts of parameters

And now?
What can we do with current hardware?
Evaluation on Current Hardware
Our implementation

- Unable to obtain Malkin et al.'s implementation
- C implementation
- Uses OpenSSL libraries for computations and communications
- Generalization of distributed sieving
- Parallelization to absorb latency
- Failure tolerance to nodes dying
Parallelization

Without parallelization

- Each peer generates $p_i$ and $q_i$
- Several synchronized rounds to obtain $N$

$\Rightarrow$ Time is spent waiting for others’ values

With parallelization

- Each peer generates several $p_i$ and $q_i$
- Several synchronized rounds to obtain several $N$

$\Rightarrow$ Average waiting time is divided by the number of threads
Failure Tolerance

In a real setup, nodes die...

- Nodes disconnect
- Nodes crash

⇒ Each bi-primality test is an independent round, program should continue

Failure tolerance

- When a node stops responding, all other return to step 1
- Every peer wait for other peers to restart
Deployment on PlanetLab

We deployed this program on PlanetLab:

- Worldwide P2P testbed
- 1,000 computers
- High usage
- Nodes die unexpectedly

⇒ Pessimistic setup with high latency, low bandwidth and overloaded CPU
Number of iterations to find $N$

Number of iterations to find 1024 bit $N$ product of two primes
3 servers on LAN

<table>
<thead>
<tr>
<th>Modulus size</th>
<th># iterations</th>
<th>Data sent</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024 bits</td>
<td>1099</td>
<td>3.1 MB</td>
<td>25s</td>
</tr>
<tr>
<td>2048 bits</td>
<td>4213</td>
<td>22.2 MB</td>
<td>3 min 43s</td>
</tr>
<tr>
<td>4096 bits</td>
<td>16227</td>
<td>166.5 MB</td>
<td>56 min 6s</td>
</tr>
</tbody>
</table>

Performance in function of the modulus size, using 3 servers on a LAN
## Comparison LAN/WAN

<table>
<thead>
<tr>
<th>Network</th>
<th>Time per iteration</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAN</td>
<td>0.07s</td>
<td>1 min 18 sec</td>
</tr>
<tr>
<td>PlanetLab</td>
<td>2.27s</td>
<td>50 min</td>
</tr>
</tbody>
</table>

Performances in function of the network (10 servers, 30 threads, 1024 bit modulus)
### Impact of parallelization

<table>
<thead>
<tr>
<th># threads</th>
<th>Time per iteration</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>0.44 s</td>
<td>8 min 3s</td>
</tr>
<tr>
<td>50</td>
<td>0.20 s</td>
<td>3 min 40s</td>
</tr>
<tr>
<td>100</td>
<td>0.15 s</td>
<td>2 min 45s</td>
</tr>
</tbody>
</table>

Effect of multi-threading with 3 servers on PlanetLab, 1024 bit modulus
Experiments on WAN

<table>
<thead>
<tr>
<th># servers</th>
<th># threads</th>
<th>Data sent</th>
<th>Time per iteration</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>30</td>
<td>39 MB</td>
<td>2.72s</td>
<td>50 min</td>
</tr>
<tr>
<td>21</td>
<td>100</td>
<td>181 MB</td>
<td>6.44s</td>
<td>118 min</td>
</tr>
<tr>
<td>37</td>
<td>300</td>
<td>572 MB</td>
<td>11.79s</td>
<td>215 min</td>
</tr>
</tbody>
</table>

Some example runs on PlanetLab with a 1024-bit modulus
Conclusion
Implementation of the Boneh and Franklin distributed RSA key generation algorithm
Tests on a large network
Keys can be generated by a few tens of peers
More peers ⇒ less trust in each peers

GPL code available at :

www.rennes.supelec.fr/ren/perso/flesueur/sgrsa.htm
Experimenting with Distributed Generation of RSA Keys

Thierry Congos, François Lesueur
firstname.lastname@supelec.fr

SUPÉLEC, SSIR Group (EA 4039), France

STM, September 24 2009
Saint-Malo, France