Experimenting with Distributed Generation of RSA Keys

Thierrry 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

$\Rightarrow$ *Certification Authorities, centralization*

Our Context: Distributed Certification

- Capacities are still proved by certificates
- These certificates are signed collaboratively by members

$\Rightarrow$ *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

$\Rightarrow$ Key must be distributedly generated !

Outline

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 $d \times e \equiv 1 \mod \varphi(N)$

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

Distributed RSA Key Generation Algorithm (Boneh and Franklin) 2/2

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

$\Rightarrow$ 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

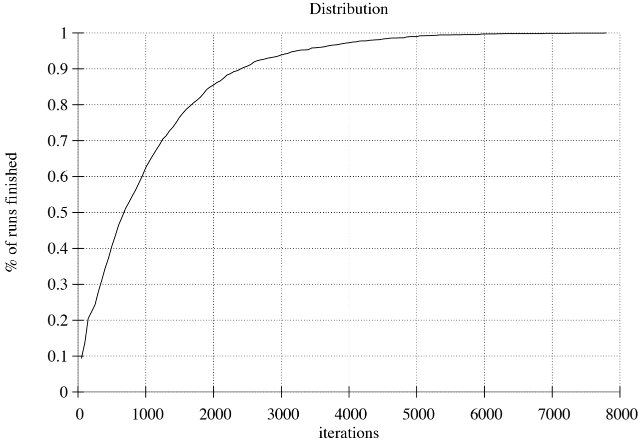
Deployment on PlanetLab

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

Modulus size	# iterations	Data sent	Total time
1024 bits	1099	3.1 MB	25s
2048 bits	4213	22.2 MB	3 min 43s
4096 bits	16227	166.5 MB	56 min 6s

Performance in function of the modulus size, using 3 servers on a LAN

Comparison LAN/WAN

Network	Time per iteration	Total time
LAN	0.07s	1 min 18 sec
PlanetLab	2.27s	50 min

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

Impact of parallelization

# threads	Time per iteration	Total time
11	0.44 s	8 min 3s
50	0.20 s	3 min 40s
100	0.15 s	2 min 45s

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

Experiments on WAN

# servers	# threads	Data sent	Time per iteration	Total time
10	30	39 MB	2.72s	50 min
21	100	181 MB	6.44s	118 min
37	300	572 MB	11.79s	215 min

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 $\Rightarrow$ 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

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

SUPÉLEC, SSIR Group (EA 4039), France

STM, September 24 2009
Saint-Malo, France