Experimenting with Shared Generation of RSA Keys

Michael Malkin

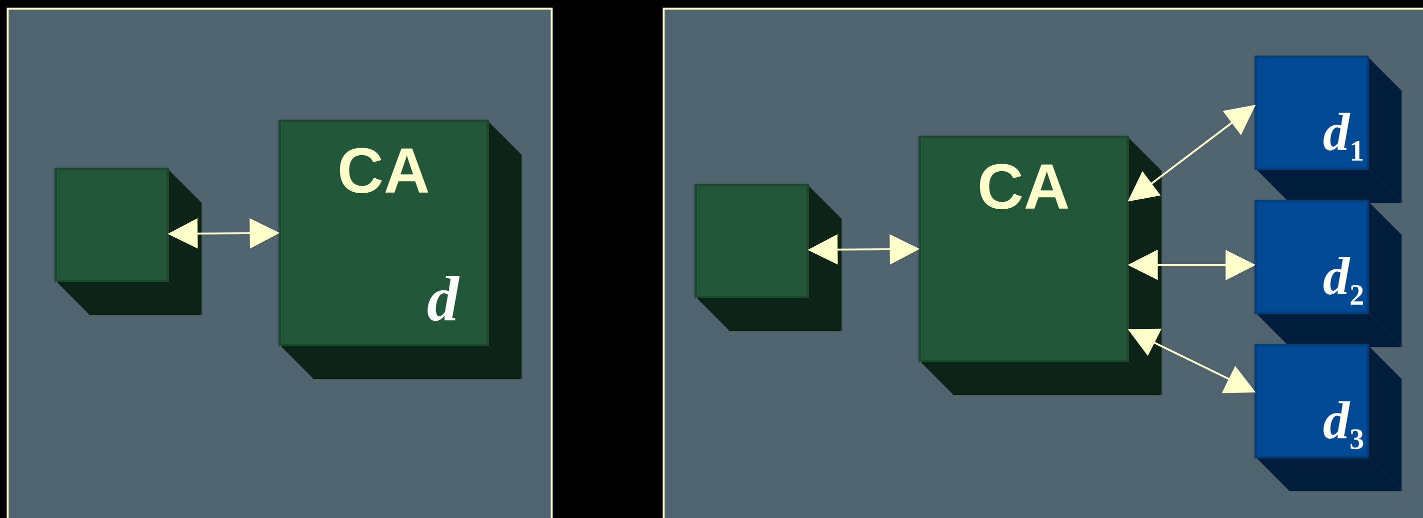
Thomas Wu

Dan Boneh

Stanford University

***Supported by DARPA**

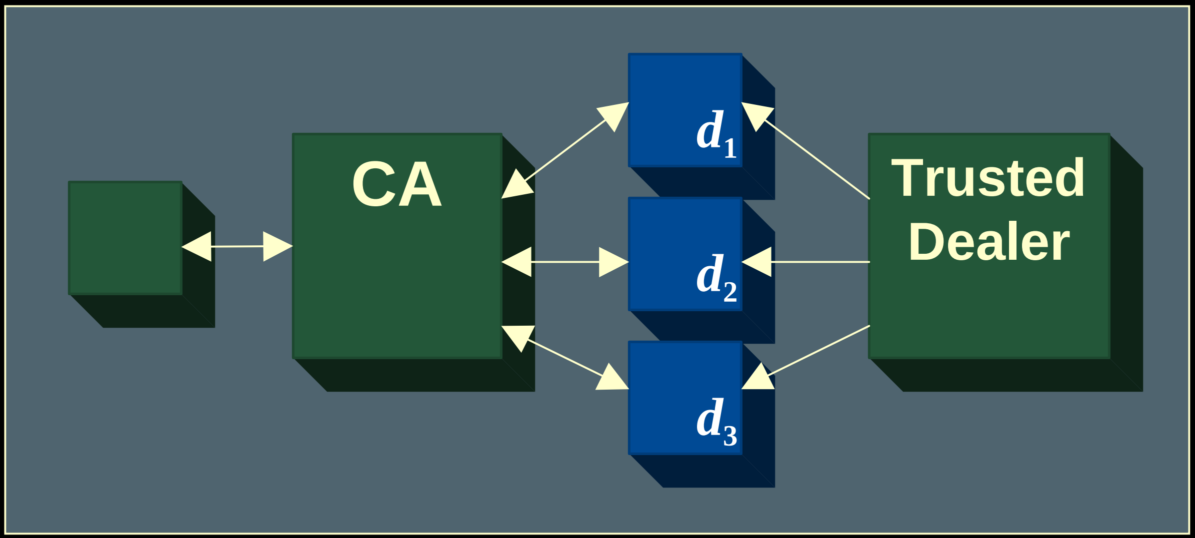
Why Share Keys?



The private key is never reconstructed!

☞ ***Who generates the shared key?***

Trusted Dealers



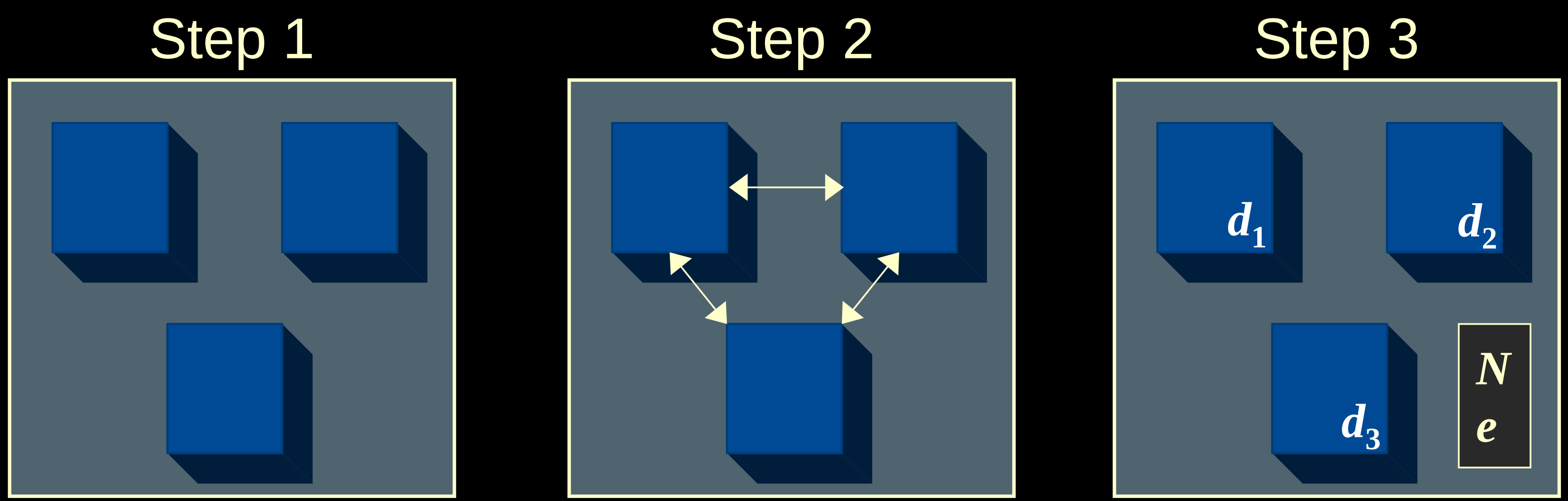
Drawbacks:

- Single point of failure
- May have to destroy dealer afterwards

Distributed Generation

Advantages:

- Nobody ever knows the entire key
- No single point of failure



RSA Keys

N	An n-bit modulus, $N = pq$
e	The encryption (public) key
d	The decryption (private) key

Sharing of d : $d = d_1 + d_2 + d_3$

☞ Can apply key without reconstructing d

d is the secret

p or $q \rightarrow d$

Distributed Generation*

1

p_1 q_1 | p_2 q_2 | p_3 q_3

3

Biprimality Test

p_i, q_i are $n/2$ bit integers

2

p_1 q_1 | p_2 q_2 | p_3 q_3 | N

4

p_1, q_1 d_1 | p_2, q_2 d_2 | p_3, q_3 d_3

$N = (p_1 + p_2 + p_3) \cdot (q_1 + q_2 + q_3) = pq$

Nobody ever knows p or q!

(*Boneh-Franklin)

How Do They Compare?

Non-Distributed:

- Pick prime p
- Pick prime q
- Multiply

Distributed:

- Pick N
- Hope $N = pq$ is an RSA modulus
- Can't test p and q separately

☞ ***Distributed generation takes more iterations***

Main Results

Initial time: 2.5 hours
(1024-bit key)

• Distributed Sieving	× 10
• Multithreading	× 6
• Load Balancing	× 1.3
• Parallel Trial Division	× 1.3

Final time: 1.5 minutes

Minding Your p's and q's

- Bad $N \rightarrow$ probably divisible by 3 or 5 or 7 or …
- Idea: Ensure that N isn't divisible by any small primes

☞ Distributed Sieving

- Can pick p_i, q_i so that p, q are not divisible by small primes

… But nobody actually knows p or q!

Using Idle Time

- Synchronous algorithm → synchronization delays
- Under-utilizing CPU — idle 80% of time

☞ Multithreading

- 6 threads optimal for 1024-bit key
- Almost 6 times faster!

(On 300Mhz Pentium II's running Solaris 2.6)

Costly Biprimality Test

- Biprimality test involves time-consuming calculation
- Idea: Only one server needs to do this

☞ Load Balancing

- A different server does test for each iteration
- Probabilistic load balancing

More Small Primes

- What about small primes not covered by sieving?
- Trial division on N by small primes

☞ Parallel Trial Division

- Each server does trial division on different small primes

Private Key Generation

- Implemented method for small e
- In RSA usually use a small e

- After N is found, generate d_1, d_2, and d_3 so:

$$d_1 + d_2 + d_3 = d$$

… But do this so that nobody ever knows d

- There is an additional way to share d
- Only a fraction of servers need to be active

Implementation: Config File

```
Num_Servers:                3
Key_Length:                 Normal
Threads:                    2

TrialDiv_End:               10000
Sieve:                      True
Test_Mode:                  True
Sequence_Numbers:           True
Transport:                  sslv3


Share_IP_Port_0:                8080
Server_IP_Addr_0:               ittc.stanford.edu
Server_Sequence_File_0:         com_security/seq0
Server_Cert_0:                  com_security/cert_s0.pem
Server_Key_0:                   com_security/key_s0.pem
```

Implementation: COM

- Abstraction layer
- Fault tolerance - non-blocking I/O
- Private, authenticated channels
  - Based on SSLeay

- Authenticates share servers using a server certificate:

```
/C=US/ST=California/O=Stanford University/
   OU=ITTC Project/CN=[SERVER 0]
```

Shared Key Storage

- Stored as PEM-encoded ASN.1 format

Data Type	Field
Integer	Version
Integer	N
Integer	e
Integer	k
Integer	d_1
⋮	⋮
Integer	d_k

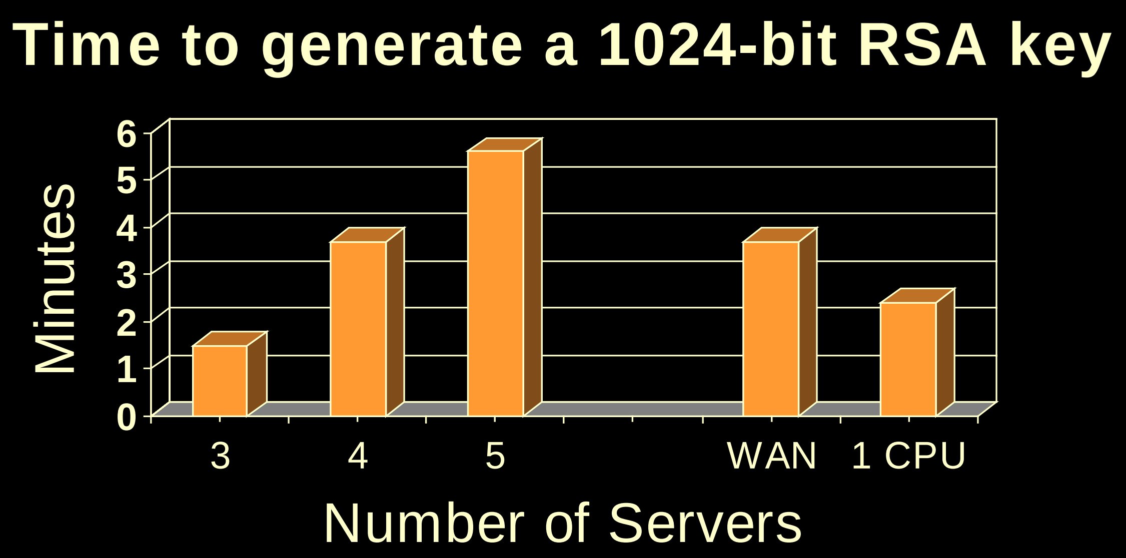
Performance

Key Size	Threads	Primality Tests	Iterations	Total Time	Network Traffic
512 bit	2	36	119	0.15 min	0.18 Mb
1024 bit	6	49	130	1.5 min	1.16 Mb
2048 bit	6	234	495	18 min	7.48 Mb

On three 300Mhz Pentium II's running Solaris 2.6

- Network bandwidth is reasonable
- 1024-bit works well
- 2048-bit is reasonable

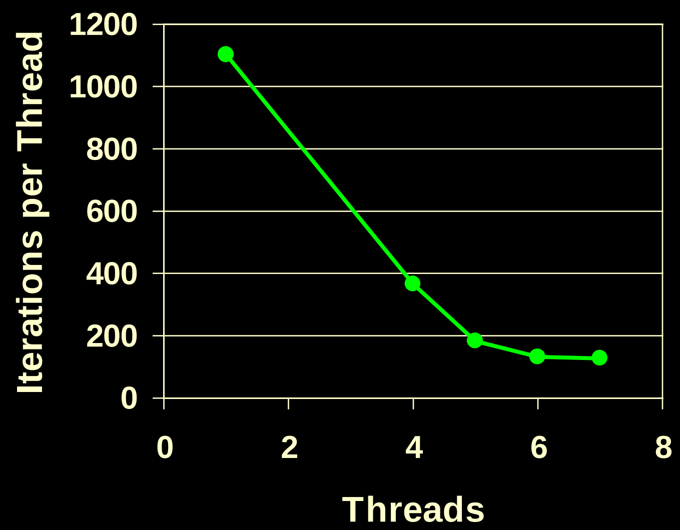
Effect of Number of Servers



WAN:

- Two servers at Stanford
- One server at University of Wisconsin at Madison
- Difficult to find PC's running Solaris

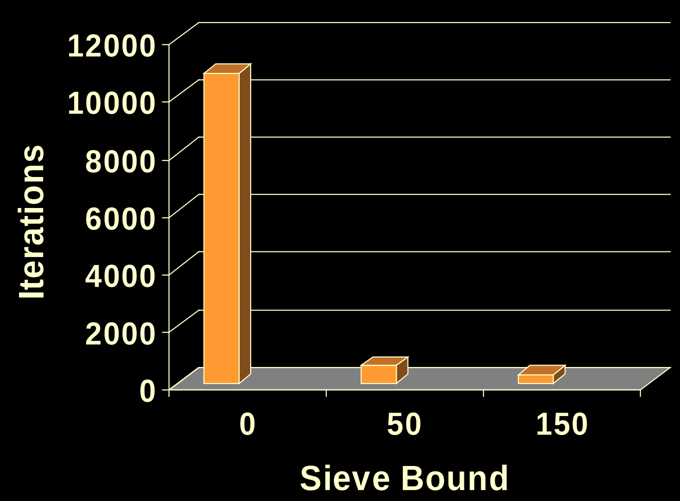
Effect of Threads

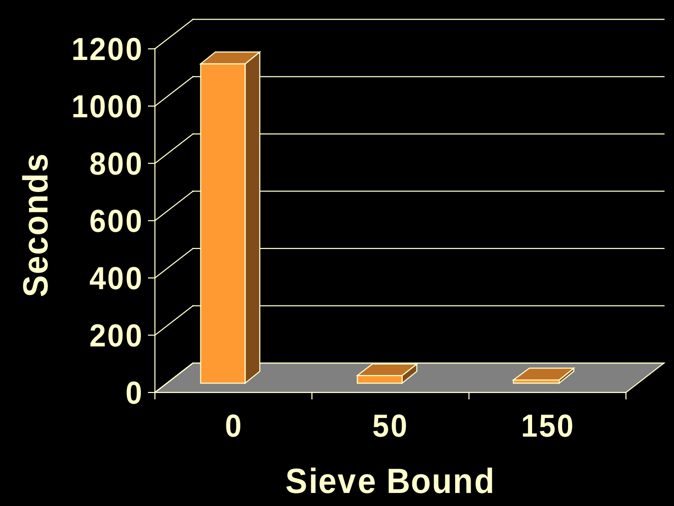


- Synchronization/CPU tradeoff
- Minimize time with 6 threads

*Generating a 1024-bit RSA key

Effect of Distributed Sieving





- Sieve bound is largest prime sieved
- Larger sieve → fewer iterations
- Diminishing returns

*Generating a 512-bit RSA key

Conclusions

☞ Distributed key generation is practical:
- 1.5 minutes for 1024-bit key

☞ Several practical improvements to algorithm
- Distributed Sieving
- Multithreading
- Load Balancing
- Parallel Trial Division

☞ Optimized cryptographic algorithm
- Requires security proofs

`http://theory.stanford.edu/~dabo/ITTC`