A Distributed Key Generation Technique

Chenxi Wang
William A. Wulf
{cw2e | wulf @ virginia.edu }
URL: http://www.cs.virginia.edu/~cw2e/kgen.ps

Abstract

In a public key cryptographic system, the uniqueness and authenticity of the keys are essential to the success of the system. Traditionally, a single, centralized key distribution/certification server has been used to generate and distribute keys. This approach requires a distinguished trusted entity which could potentially become a single point of failure or penetration in a distributed environment. We present in this paper a new, simple way to handle distributed key generation. We assign a unique range of m-bit numbers to each key generator in the system. As a result, the lower-order m bits of the keys generated is a unique number in the assigned range. Our scheme not only provides a way to generate globally unique keys in an independent, distributed fashion, it also enhances the security of public-key cryptosystems by eliminating the mapping between keys and entity names.

1 Introduction

The authors are participating in a project construct a large distributed computing environment called Legion. The details of this system are mostly irrelevant to the content of this paper, except that three aspects of it provide the motivation for the problem of a distributed, unique key generation system.

First, Legion is intended to be large, literally involving millions of computers distributed across the world. Thus all techniques used in its construction must be “scalable” -- that is, we cannot tolerate situations in which a small finite number of resources or servers could be either bottlenecks or points of failure.

Second, Legion is an object-oriented system; all resources are conceptually modeled as objects whose methods provide services. Among other things, this implies that every object must have a globally-unique name. Mindful of the first point above, that means we need a means of scalable, distributed unique name generation.

Finally, security is a principal design goal of Legion. The details of the Legion security model are irrelevant here, but they include the absolute necessity of authentication. Before any security decision can be made we must be sure of who is requesting what service on which object. We have decided to base our authentication on the use of public key cryptography.

With these attributes in mind we made another critical decision -- namely to make the name of an object be its public key! That is, the bit string that is the globally unique name is also the bit string that is its public key!

This decision eliminates one potential bottleneck and source of security concern, namely the need for a trusted server that provides/certifies the mapping from name to key. Alas, it creates another problem, namely the topic of this paper -- the need for scalable, distributed unique key generation.

Without this decision there would still be the need for unique name generation, but there would have been no need for the name to be a suitable key. Similarly, we would have still needed a distributed key generator, but the keys would not have had to be unique -- a sufficiently low probability of two keys being the same would have been good enough.

2 Background

In this section we review the basic concepts of public-key cryptography and RSA algorithm which we will need at our fingertips for later discussion.

2.1 Public-key Cryptography

In a public-key cryptosystem, each principle has a pair of public/private keys. The public key of a principle is made known to the world while he/she keeps the corresponding private key secret. A public-key cryptosystem has the following properties (E stands for the encryption procedure and D stands for the decryption procedure):

- Deciphering the enciphered form of a message M yields M. Formally

  D(E(M)) = M
- Both E and D are easy to compute
- By publicly revealing E the agent does not reveal an easy way to compute D.
- If a message M is first deciphered and then enciphered, M is the result. Formally,

  E(D(M)) = M

When a principle publishes his public key, he reveals no easy way of computing the corresponding private key. The security of a public-key cryptosystem rests both on the difficulty of recovering the private key from its public counterpart and the difficulty of deducing the plaintext from the ciphertext.

2.2 RSA

In 1978, R. Rivest, A. Shamir, and L. Adleman proposed the first full-fledged public-key algorithm[2]. Now known as RSA, it is one of the few proposed public-key algorithms that are both secure and practical. Capable for both encryption and digital signatures, RSA is the best-known and most widely-used public-key algorithm.

RSA relies its security on the difficulty of factoring large numbers. The public and private keys are functions of a pair of large prime numbers. It is conjectured that recovering the private key or the plaintext of any encrypted message is equivalent to factoring the product of the two primes.

The first step to generate RSA keys is to randomly choose two large prime numbers, p and q. Compute the product:

$$N = p \bullet q$$

N is called the modulus. Next is to generate the encryption key *e* and decryption key *d*. *e* is a randomly selected number such that *e* and $(p-1) \cdot (q-1)$ are relatively prime:

$$(e,\ (p-1)\cdot(q-1)) = 1$$

The decryption key d is then computed as e’s inverse modulo (p-1)(q-1):

$$ed \equiv 1\ (mod\ (p-1)\ (q-1))$$

The public key is the pair of numbers — (*N*, *e*). *d* is the private key. The two prime numbers *p* and *q* are no longer needed. They should be discarded, but never revealed.

Having established the keys, encryption is accomplished by raising the message block *M* to its *e*-th power and performing a modulo *N* operation on the exponentiation result:

$C = E(M) = M^e\ mod\ N$, where *C* is the ciphertext block

Message M is recovered by applying a similar operation to the cyphertext block, this time is with the decryption key *d*, again modulo *N*.

$$M = D(C) = C^d\ mod\ N$$

Reversing the above steps provides the signing and verification of digital signatures.

3 The proposed key generation technique

We introduce in this section a distributed key generation scheme in a RSA cryptosystem, as well as some mathematical background to support this work

3.1 Number Theory Theorems

Consider a general linear congruence

$$k \cdot x \equiv l\ \textbf{\textit{mod c}}$$

Theorem 1: Let (k, c) = d, the linear congruence is solvable if and only if $d|l$. It has then infinitely many solutions but only d incongruent solutions.

The proof of this theorem is out of the scope of this paper. Readers should refer to Hardy &. Wright [1] or J. Strayer[2]

Theorem 2: Given an m-bit number R, and an odd m-bit number p, there exists an unique m-bit integer q, also odd, such that,

$$p \cdot q \equiv R\ (\text{mod } 2^m)$$

Proof: Proof of this theorem follows directly from Theorem 1.

d = (p, 2^m) = 1, therefore d | R. The linear congruence is solvable. Although there are infinitely many solutions to the congruence modulo 2^m, there is exactly one incongruent solution q such that 0 < q < 2^m.

Solving this linear congruence requires the use of Euler’s Theorem. The solutions can be expressed in the following form:

$$q \equiv \left(R \cdot p^{\phi(2^m)-1}\right) \text{mod } 2^m.$$

3.2 Generating Keys

Our goal is to design a key generation scheme that generates globally unique keys in a completely independent and distributed fashion. The merit to distributed key generation is self-manifest: No distinguished trusted entity is required. However, how to generate UNIQUE keys poses a challenging problem. The emphasis is on “unique” because it is absolutely crucial that the same keys never be generated twice; each must be unique! Therefore, we first need to devise a way to ensure the uniqueness of the keys.

Our approach to the problem is to assign each key generator a unique range of m-bit numbers. In Legion, these numbers contain the object’s class identifier and instance identifier. The purpose is to have some property of the keys generated fall in the assigned range of their generators. To be more specific, assume that the i-th key generator is assigned the range [Li, Ui], inclusive, the task of the key generation is to generate keys whose lowest m bits fall in the assigned range. More formally,

$$Li \le K\left(mod 2^m \right) \le Ui \text{, , where K is the key}$$

Equally important is the distinction that no single key generator is assumed trusted by everyone. Each is trusted by some users but not necessarily by others.

Without loss of security, we assume that, in a RSA cryptosystem, the encryption key *e* is common for every entity in the system while *N* is unique. Therefore only the modulus part of the public keys need to be generated. In the rest of the text, we will refer to the public modulus as the public key.

3.2.1 The Key Generation Algorithm

This subsection presents an algorithm that implements the above idea. The algorithm itself is very simple and straightforward: At each run of the algorithm, a seed number *R* is selected from its assigned range and targeted as the resulting public key’s lowest m bits. In other words,

$$N \,(\mathrm{mod}\ 2^m) = R$$

The keys generated must satisfy the constraints of the RSA algorithm. That is, the public key must be the product of two prime numbers. Note that both public and private keys are functions of the two primes. Our main task thus is to find two suitable prime numbers *p* and *q* whose product has R as its targeted lowest m bits:

$$p \cdot q \ \ (mod\ 2^m) = R$$

Supposing n is the bit length of the public key, p, q are both n/2 bits long and the seed number R is m bits long. The results of theorem1 and theorem 2 tell us that when m is exactly n/2, we can uniquely determine q based on p and R such that

$$(N = p \cdot q) \ \mathrm{mod} \ 2^m \ = R$$

Now we can detail out the actual algorithm:

1. Select a seed number R from the assigned range

2. if m < n/2, generate a random number R’ of (n/2 - m) bits long. Assign ($\left(R' \cdot 2^m \right)$ + R to R.

3. Randomly select a prime number p of n/2 bits

4. Compute $q = \left(R \cdot p^{\phi(2^m) - 1}\right) \bmod 2^m$

5. Run primality test on q. If q is prime, then we are done. Otherwise repeat steps 3 - 5 until a pair of suitable prime numbers, p and q, are found.

4 Analysis

In this section we will discuss the complexity of our algorithm and cryptographic strength of the keys generated using this new method. We then compare our scheme with the conventional methods and discuss the advantages of this new scheme.

4.1 Mathematical analysis

The method we proposed is a probablistically unbiased process for generating p, q based on the targeted m-bit number. Admittedly, this scheme is more complex and takes longer to execute than the regular RSA key generation algorithm, but we believe that the ability of doing distributed, unique key generation far outweighs the increased complexity.

The prime number theorem says: The number of prime numbers less or equal to any number x is asymptotically close to

$\left(\frac{x}{\ln(x)}\right)$, (see Harvy & Wright [7])

So, on average, one could expect to search approximately ln(x) numbers before a prime number less or equal to x is found.

Armed with the result of this theorem, we estimate the probability of q, once p and R are determined, being prime is approximately $\frac{2}{m}$ (note that we only consider odd numbers) where m the bit length of q. Assuming p and q are of equal length, we can now estimate the complexity of our algorithm — on average, $\frac{m^2}{4}$ numbers are searched before a pair of p and q are found.

This estimate is only a rough measure of the algorithm complexity because it is reached under the assumption that p and q are picked independently while in our algorithm the value of q is determined by p and R. Table 2 gives sample measures of the speed of our algorithm. The speed is measured in the numbers of primality testings(PTs) needed before a pair of good primes, p and q, are found. Figure 2 gives a comparison between the estimate and the sampling results in Table 2.

An interesting question to ask is whether such a pair of primes p and q always exists for any given R. To answer this, we have the following yet to be proven conjecture:

Conjecture 1: For any given seed number R of n/2 bits long, there always exists at least one pair of primes p and q such that $p \cdot q \equiv R \bmod 2^m$ .where p and q are both of n/2 bits long.

	64-bit key	128-bit key	256-bit key	512-bit key	1024-bit key
R1	525 PTs	1576 PTs	5154 PTs	21629 PTs	98353 PTs
R2	365 PTs	1666 PTs	4915 PTs	23865 PTs	50954 PTs
R3	463 PTs	1476 PTs	4900 PTs	19994 PTs	62848 PTs
R4	388 PTs	1794 PTs	4994 PTs	16000 PTs	88310 PTs
R5	553 PTs	1428 PTs	4438 PTs	18026 PTs	63320 PTs
R6	463 PTs	1306 PTs	6134 PTs	18604 PTs	69335 PTs
R7	363 PTs	1320 PTs	5467 PTs	19564 PTs	71230 PTs
R8	426 PTs	1582 PTs	6242 PTs	26395 PTs	65569 PTs
Average	443 PTs	1518 PTs	5280 PTs	20142 PTs	71239 PTs

Table 1: Sample Running Results

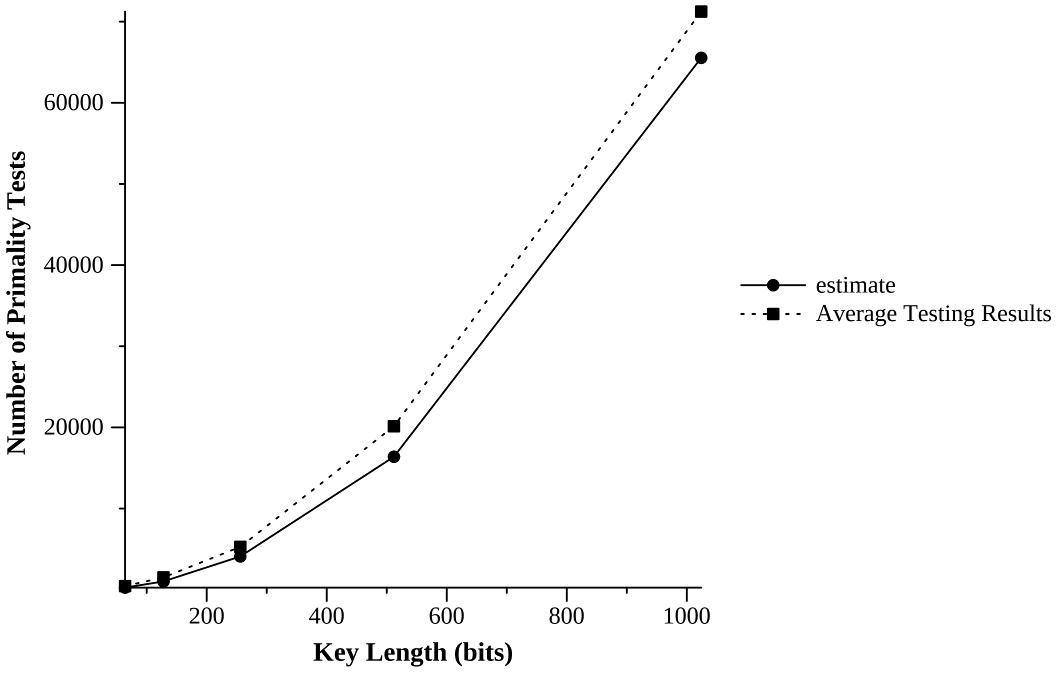

Figure 2

4.2 Security Analysis

The next logical question is the cryptographic strength of the keys. We have an intuitive argument in which we conjecture that the keys generated by our method is at least as strong as the keys of the same length generated by a regular RSA key generation algorithm.

The argument is rather simple: In comparison to RSA, we added one more constraint to the key generation process. That is, the lower m bits of the resulting keys lie in some known range. This constraint reveals no useful information that an adversary can utilize to factor N. Once the public key (N, e) is publicized, everyone knows the lowest m bits of N. And everyone knows the fact that the two secret primes p and q, when multiplied, must produce these lowest m bits. The adversary simply cannot use this fact to his advantage. Breaking the keys still requires solving of the factoring problem. The keys generated using our algorithm is, cryptographically speaking, as strong as the regular RSA keys.

5 Conclusion

Conventional public-key cryptosystems often rely on a centralized key distribution/certification server. Use of such centralized components in distributed systems does not scale well. The potential size and scope of today's large networking systems suggest that we simply could not have distinguished "trusted" entities that could become a single point of failure or a system bottleneck.

We presented in this paper a distributed key generation technique that handles key distribution in a completely distributed and independent manner. The proposed method requires no centralized entity and enables individual key generators to generate globally unique keys without consulting a central authority or each other. We then demonstrated an implementation of the algorithm based on RSA. Finally, we analyzed the complexity and security implication of this new method.

Comparing with conventional key generation methods, our scheme enables distributed generation of globally unique keys. It scales well in the context of large distributed systems. Our initial motivation for this work is to eliminate centralized components in Legion. In the future, we plan to develop similar schemes in other cryptosystems to achieve a broader level of applicability.

6 Reference

1. [Diff76] W. Diffie and M. Hellman. New directions in cryptography. IEEE Trans. Inform. Theory IT-22, 6 Nov. 1976, 644-654
2. [RSA78] R. L. Rivest, A. Shamir, and L. Adleman, A Method for Obtaining digital Signatures and Public-key Cryptosystems, Communication of ACM, Vol 21, No. 2, 1978
3. [RSA95] RSA laboratories Seminar, RSA Key Generation and Strong Primes, June 1995
4. [FACT] David M. Bressoud, Factorization and Primality Testing, Springer-Verlag
5. [CRYP96] Bruce Schneier, Applied Cryptography, John Wiley & Sons, Inc.
6. [PRIM] Evangelos Kranakis, Primality and Cryptography, Wiley-Teubner Series in Computer Science
7. [HARVY68] G. H. Hardy, E. M. Wright, The Theory of Numbers, Oxford at the clarendon Press, 1968