VOLUME 3, NUMBER 1 — SPRING 1997

RSA LABORATORIES'

CryptoBytes

The technical newsletter of RSA Laboratories, a division of RSA Data Security, Inc.

Contents

1
Proactive Security: Long-term Protection Against Break-ins

2
Editor's Note

9
Fast Generation of Random, Strong RSA Primes

14
Algorithms Update

15
Standards Update

16
Announcements

Proactive Security: Long-term Protection Against Break-ins

Ran Canetti
Rosario Gennaro
IBM T.J. Watson Research Center

Amir Herzberg
Dalit Naor
IBM Haifa Research Center

Introduction

Cryptography offers a set of sophisticated security tools for a variety of problems, from protecting data secrecy, through authenticating information and parties, to more complex multi-party security goals. Yet, the most common attacks on cryptographic security mechanisms are 'system attacks' where the cryptographic keys are directly exposed, rather than cryptoanalytical attacks (e.g., by analyzing ciphertexts). Such 'system attacks' are done by intruders (hackers, or through software trapdoors using viruses or Trojan horses), or by corrupted insiders. Unfortunately, such attacks are very common and frequently quite easy to perform, especially since many existing environments and operating systems are insecure (in particular Windows).

As a result, computer and network security involve a set of ad-hoc tools to prevent and detect intrusions, and to regain control over a computer from the attacker. Detection is particularly important, since once an attack is detected on any one computer, system administrators are alarmed and are likely to regain control from the attacker — on most or all computers. Furthermore security measures are likely to be tightened, and at least some security exposures found and fixed. Therefore, attackers often do their best to avoid detection, and indeed often give up control over a computer rather than risk being detected.

A common approach to enhancing the security is *periodic refreshments* of secrets. Examples include refreshments of passwords, and of session-key refreshment in secure communication protocols (such as IP-SEC [5] and SSL/TLS). The idea is to make 'old secrets' (i.e., secrets from before the refreshment) useless for the attacker. Thus the attacker is forced to either lose control or to be constantly active, thus risking detection.

Another approach to enhancing the security is the *distribution* of cryptographic trust among several components, or servers. This approach is exemplified in secret sharing algorithms [26,3], and taken to a much greater extent in the notion of threshold cryptography [11,17]. Here a secret key is split into shares, and each share is given to one of a group of servers. The servers engage in a protocol that 'emulates' the behavior of the centralized solution (the case where the key is kept in one piece). The protocol ensures security as long as at most some predefined number (a 'thresh-

(continued on page 3)

Ran Canetti and Rosario Gennaro are Research Staff Members at the IBM T.J. Watson Research Center. They can be contacted at canetti@watson.ibm.com and rosario@watson.ibm.com, respectively.
Amir Herzberg manages the Network Computing and Security Group at the IBM Haifa Research Lab (Tel-Aviv Annex), and Dalit Naor is a Research Staff Member there. They can be reached at amir@haifa.vnet.ibm.com and dalit@haifa.vnet.ibm.com.

RSA Laboratories
A Division of RSA Data Security

Editor's Note

This issue of *CryptoBytes* marks the start of the third volume. With a world-wide readership of more than 6,000 it has rapidly become a useful forum for cryptographic updates and for presenting some of the latest research results.

We encourage any readers with comments, opposite opinions, suggestions or proposals for future issues to contact the CryptoBytes editor.

One recent area of cryptographic research has been that of *proactive security*. In the field of computer and network security, systems might come under attack as adversaries attempt to control part of a network. The task facing the administrator of such networks is to detect and reverse any such compromises. Among the techniques available are so called threshold techniques (surveyed in the last issue of *CryptoBytes*) with which the storage and use of sensitive cryptographic information can be shared. This forces an attacker to undertake far more work in attempting to compromise the system. The proactive techniques described here by Ran Canetti, Rosario Gennaro, Amir Herzberg and Dalit Naor can add an additional dimension to this protection. Using such techniques it is possible to automatically recover from undetected break-ins, and to force an attacker to restart the process of compromising the system from scratch.

In the second article of this issue, Bob Silverman presents the recommendations he made to the X9.31 standards committee on the generation of prime numbers when using RSA. The relevance of so-called "strong" primes to the security of the RSA cryptosystem has long been open to question and the position of RSA Laboratories is that such primes offer little, if any, additional practical protection to the user of RSA. However, some standards bodies have adopted calls for RSA moduli that are composed of such primes and here Bob Silverman describes one way of generating them.

As always, the newsletter contains some of the latest news from the world of algorithm and standards development. We report on the solving of the DES challenge by a distributed search effort and also on the publication of a description of RC2® as a part of the S/MIME standardization effort. In the standards arena, we report on the latest developments in the Public Key Cryptography Standards. The PKCS have become widely used in the cryptographic industry. However, over the years there have been substantial advances in cryptographic knowledge and algorithm design. It is now time for the PKCS suite to undergo revision, and this substantial undertaking has been started in RSA Laboratories.

The future success of *CryptoBytes* depends on input from all sectors of the cryptographic community, and as usual we would very much like to thank the writers who have contributed to this first issue of the third volume. We encourage any readers with comments, opposite opinions, suggestions or proposals for future issues to contact the *CryptoBytes* editor at RSA Laboratories or by E-mail to *bytes-ed@rsa.com*.

Newsletter Availability and Contact Information

CryptoBytes is a free publication and all issues, both current and past, are available via the World-Wide Web at <*http://www.rsa.com/rsalabs/pubs/cryptobytes.html*>.

For each issue a limited number of copies are printed. They are distributed at major conferences and through direct mailing. While available, additional copies of the newsletter can be requested by contacting RSA Laboratories though a nominal fee to cover handling costs might be charged for individual requests.

RSA Laboratories can be contacted at:

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065
415/595-7703
415/595-4126 (fax)
rsa-labs@rsa.com

About RSA Laboratories

An academic environment within a commercial organization, RSA Laboratories is the research and consulting division of RSA Data Security, the company founded by the inventors of the RSA public-key cryptosystem. Its purpose is to provide state-of-the-art expertise on cryptography and information security for the benefit of RSA Data Security and its customers. RSA Data Security is a Security Dynamics company.

Continued from page 1

old') of servers are broken into. Threshold cryptography can indeed enhance the security against break-in attacks in many scenarios. However, it is also limited: Given sufficient amount of time, an attacker can break into the servers one by one, thus eventually compromising the security of the system. This danger is particularly eminent in systems that must remain secure for long periods of time (such as certification authorities) or where secure recovery may be difficult (such as with secure communication).

Proactive security is a mechanism for protecting against such long-term attacks. It combines the approach calling for distribution of trust with the one of periodic refreshment:

Proactive = Distributed + Refresh

That is, first distribute the cryptographic capabilities among several servers. Next, have the servers periodically engage in a **refreshment protocol**. This protocol will allow servers to automatically recover from possible, undetected break-ins, and in particular will provide the servers with new shares of the sensitive data while keeping the sensitive data unmodified. Very importantly, information gathered by an attacker before a refreshment period becomes useless to attack the system in the future. We will see later that sometimes the attacker may be able to prevent complete automatic recovery of system security, however in these cases the attack is detected beyond doubt. As explained above, this would enable highly effective (yet expensive) manual security measures to be applied, and security would ultimately be restored. In all, the security of the system will be guaranteed as long as not too many (say, less than half) of the servers are broken into *between two consecutive executions of the refreshment protocol*. Notice that in this approach, security is preserved even if, over a long span of time, every server can be broken into at some time or another. In other words, a proactively secure system does not wait until a break-in is detected. Instead, it invokes the refreshment protocol periodically (and proactively) in order to maintain uninterrupted security, or force detection.

Before proceeding any further, let us present an example where proactive security seems very much called for: **Certification authorities** (CA's). Such systems must remain secure for very long periods of time. Moreover, the security of all clients of the CA hinges on the secrecy of the CA's signing key. Thus, CA's will pose an attractive target for break-ins. A proactive solution for CA's will have the signing key shared among several servers. Signatures will be generated via a special protocol run by the servers. Furthermore, periodically (say, every day) the servers will engage in a refreshment protocol, guaranteeing the security of the CA as long as not too many (say, less than half) of the servers are broken into during the same day.

[...] in this approach, security is preserved even if, over a long span of time, every server can be broken into at some time or another.

Works on proactive security

Ostrovsky and Yung showed how a large class of multiparty protocol problems can be solved in a proactive way, in a setting where secure communication channels are available [24]. Their solution, based on the general paradigm of multiparty computation [27,19,2,6], is of significant theoretical interest but leaves the door open to efficient, practical solutions to specific problems.

In [8] the proactive approach as a security enhancement to centralized systems is considered, and a practical proactive pseudo-random generator with applications to secure sign-on is presented. Another basic task that has been 'proactivized' is secret sharing, and in particular *verifiable* secret sharing (i.e., secret sharing resilient against malicious faults) [22]. This algorithm plays a key role in proactive solutions for public-key cryptosystems, and in particular in proactive signature systems [21] (extending the threshold signature of [11]). Proactive solutions were found for the DSS signature algorithm [18,21] and for RSA [15,14].

[...] a proactively secure system does not wait until a break-in is detected. Instead, it invokes the refreshment protocol periodically (and proactively) in order to maintain uninterrupted security, or force detection.

Proactive signatures are a powerful tool. They are used in [9] to provide a proactive, automated solution to key refresh. Namely, [9] shows how to use cryptography to ensure authenticated and secret communication among servers, with recovery from penetrations and key exposures. This provides an alternative to manual key refresh. Some of the solutions mentioned above are described in more detail later in the article.

In a related vein, proactive protocols for byzantine agreement were presented in [25,16].

Applications

Proactive security has a wide range of applications. In general, proactivization can help in any scenario where the security of a system has to be maintained for a relatively long period of time. The important application to certification authorities was described above. Other applications of proactive signature algorithms include electronic cash, where the signature of the bank validates an electronic bill. Also here, the signature system has to remain secure for a long period of time.

In other applications it is beneficial to proactivize the capability to *decrypt* a message encrypted with a public-key encryption system. Examples include electronic voting systems where voters encrypt their votes with the voting center's public key [10], and secure repositories where users keep sensitive data in a certified way.

To maintain the security of secret sharing schemes even in the presence of attackers that can eventually break into all servers [...] have the servers periodically (say, every day) refresh their shares of the secret.

Implementations

We are aware of three implementation efforts of proactive security systems currently under way: by Sandia National Labs, IBM Research, and a DARPA project. We elaborate on these efforts in *Architectural design and implementations* later.

Algorithmic results

We now describe briefly three proactive techniques, which we believe to be applicable to many systems and scenarios. These are: proactive secret sharing, proactive signatures, and proactive secure communication.

Proactive secret sharing

Secret Sharing protocols were introduced in [26,3] to protect the secrecy of a value by distributing it over several servers. Typical implementation use threshold schemes in which the secret is shared among n servers and any $t + 1$ out of them can recover the secret.

To maintain the security of secret sharing schemes even in the presence of attackers that can eventually break into all servers, but can only break into a limited number of servers given limited time, have the servers periodically (say, every day) *refresh* their shares of the secret. The refreshment protocol should guarantee that the new shares are *independent of the old shares*, except for the fact that they *define the same secret*.

For example in Shamir's scheme if the secret is a value s in the set of integers $[0 \ldots p-1]$ where p is a prime, then this process can be carried out as follows [24,22]. The dealer (who is sharing the secret) generates t random numbers $a_1, \ldots, a_t$ modulo p. Given the polynomial $f(X) = s + a_1 X + \ldots + a_t X^t$ the dealer gives to server i the *share* $s_i = f(i) \bmod p$. It's clear that any t servers have no information about s while $t + 1$ can reconstruct the value by polynomial interpolation.

Periodic refreshments of the shares can be performed as follows. Each server i chooses a random t-degree polynomial $f_i(X)$ such that $f_i(0) = 0$. Server i then sends to server j the value $s_{ij} = f_i(j) \bmod p$. Server j then computes its new refreshed share $\hat{s}_j$ as follows:

$$\hat{s}_j = s_j + s_{1j} + \ldots + s_{nj} \bmod p$$

and *erases* its old share. It's easy to see that the new shares $\hat{s}_i$ lie on the polynomial $\hat{f}(X) = f(X) + f_1(X) + \ldots + f_n(X)$ which is still of degree t and whose free term is still s.

The above procedure works only in the case of a passive adversary who may read the content of memory but not modify it or cause the behavior of a server to change. In case of an active adversary the above techniques were extended using *Verifiable Secret Sharing* (VSS) protocols [7]. In particular, the VSS protocol by Feldman [13] proved especially suitable for this purpose, and in addition provides the ability to recover lost or corrupted shares and to reinstall them. See [22] for more details.

Proactive signatures

The security of public key cryptosystems relies heavily on the secrecy and integrity of the private key. Thus such cryptosystems should be augmented with methods for protecting the secret key while providing continuous availability of the system (e.g., signing capabilities).

A naive solution may be to share the private key using a proactive secret sharing scheme. This solution provides the necessary protection as long as the key is not used. However, in order to generate a signature the private key would need to be reconstructed in a single site, thus losing the advantage of distribution: A single break-in to this site will com-

promise the security. Instead, a proactive threshold signature scheme allows the servers (i.e., the shareholders) to jointly generate valid signatures in a special way that prevents an attacker from generating fake signatures. In particular, the scheme makes sure that the key is never reconstructed at a single site.

A proactive signature scheme involves three phases: the *key generation* phase (preferably done without a trusted dealer), the *joint signature-generation* phase and finally a special *proactive key refreshment* phase of the servers' key shares which is done periodically. The signature is generated in a distributed fashion from the shares of the key. Moreover, it has to hold that despite proactivization of the signing key, the signature on a message m, computed under any of the representations of the key is the same. The scheme withstands attackers that eventually break into all servers, as long as only a limited number (say, half) of the servers are broken into between two consecutive invocation of the refreshment protocol. Proactive solutions for various signature schemes have been devised, among them is a solution to RSA signatures and to DSS (Digital Signature Standard) signatures. See [21,18,15,14] for more details.

In order to exemplify an actual proactive signature scheme, we outline in the Appendix the DSS solution. It is based on the threshold signature scheme of [18] combined with [21].

Proactive secure communication

Another cryptographic task where the proactive approach seems called for is maintaining authenticated and secret communication among a set of parties. Here the parties must keep the integrity and secrecy of the relevant keys: shared keys (such as session keys), private signature and decryption keys, as well as the integrity of public keys (of other parties).

It is a standard practice to keep two levels of keys: short-lived 'session' keys, and long-lived 'master' keys. The 'master' keys are used to periodically refresh the 'session' keys (e.g. in TLS/SSL). This provides recovery from exposure of the session keys — but not of the master keys.

Protecting against the exposure of the master keys is considered a hard problem; when deemed necessary, it is achieved via manual master key refresh process, done periodically but infrequently. Some mechanisms, most notably *perfect forward secrecy* [12] (e.g., implemented by the IP-SEC standard [5]) provide protection of *past* session keys from a *future* exposure of the master keys. However, this does not protect future session keys from active impersonation attacks. Proactive security provides a more complete solution, where exposing a master key does not reveal either future or past session keys even from active attackers — achieving the same effect as the manual key refresh process, at much lower costs.

A solution may seem straightforward at first: at each refreshment phase, each party will choose a new pair of public and private keys, distribute the new public key to other parties (signed using the old secret key), receive new public keys (signed) from each other party, and then use the new public keys to agree on new shared keys. However, an intruder who also controls the communication links can successfully *impersonate* an attacked party by sending a fake public key on its behalf. Moreover, if the attacker broke into two machines, it can select fake public keys for both of them and thereby permanently 'insert' itself between the two parties. This way the attacked parties lose their ability to authenticate each other, *even long after the intruder lost control of the machines*.

These and additional problems are solved using the following idea [9]: The parties will hold shares of a proactive signature scheme (such as the ones described under *Proactive signatures*). The corresponding verification key will be held by all parties in an unmodifiable memory (e.g., a ROM or a privileged memory address). Next, in each refreshment phase, each party will obtain a *certificate*, signed jointly by the parties using the proactive signature scheme, for the authenticity of its newly chosen public keys. (The certificate may read: `the public key of party` *`P`* `at phase` *`t`* `is` …) These certificates will be used by the parties to authenticate the newly received public keys of other parties.

We remark that the above sketch is very rough; it omits many important details, to be found at [9]. (For instance, one has to modify the proactive signature scheme to withstand unauthenticated channels in the first place.)

Another cryptographic task where the proactive approach seems called for is maintaining authenticated and secret communication among a set of parties.

Proactive security provides a more complete solution, where exposing a master key does not reveal either future or past session keys even from active attackers [...]

Architectural design and implementations

The security of a proactive solution relies heavily upon its correct architecture and integration with the existing, non-proactive, operating system. We outline a plausible architectural design of a proactive system (this is the design of IBM Research's prototype). The design does not view the proactive model as series of protocols but, rather, as a security enhancement of the operating system which transforms it into a **proactively secured system**. The design also supports proactive implementation of various mechanisms such as encryption, signatures, database maintenance and other multiparty computations.

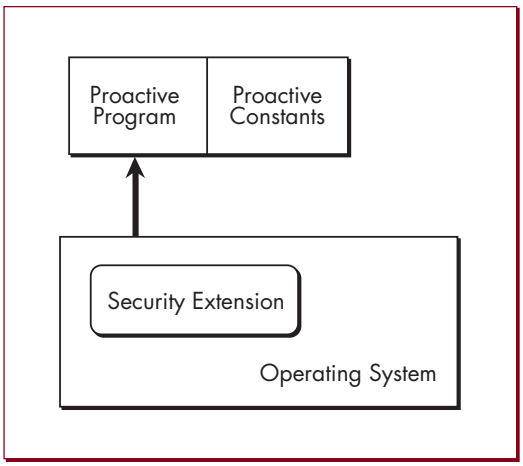

Figure 1: Operating Systems View

Figure 1 schematically depicts the integration with the operating system. The **proactive extension** consists of two parts: the proactive program and its constants. The underlying assumption is that *the program and its constants are protected against any malicious manipulation*; the rest of the memory space may be subject to any type of attack. This requirement must be addressed by any implementation that adopts the suggested architecture. It may be implemented either by some special hardware, or by an extension of the operating system's kernel to support this security requirement.

We are aware of three implementation efforts that are currently under way.

The **proactive program** should first provide a toolkit consisting of communication and cryptographic primitives which are needed to implement any proactive algorithm. In addition, it should be able to support multiple instances of proactive applications running concurrently. An essential component of such a program is a module responsible for refreshing the on-going proactive tasks of the system. A pictorial view of a plausible proactive program, which supports proactive signatures and proactive secure communication, is depicted in Figure 2.

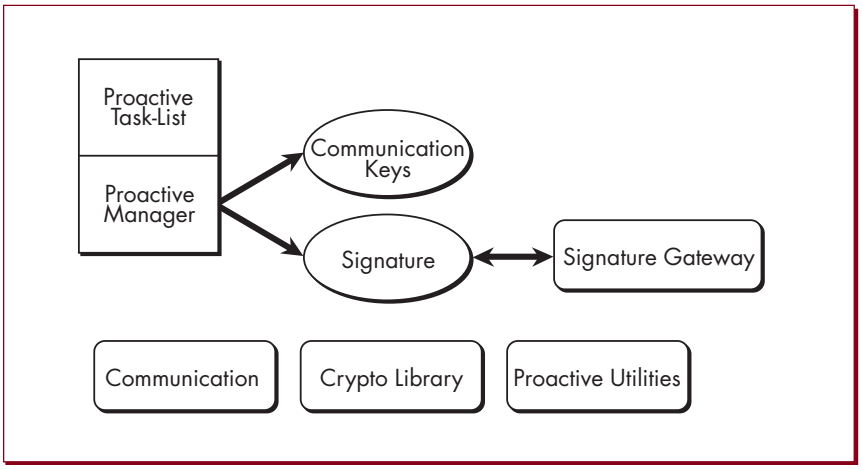

Figure 2: Architecture Model

Implementations

We are aware of three implementation efforts that are currently under way. In IBM Research, we implement a proactive-security enhancement application based on the architecture described above [23]. Aside from the basic modules, it is intended to initially implement a proactive DSS Signature Module based on [18,21], and a secure communication module based on [9]. This application could be used to enhance general security management systems. Sandia National Laboratories [20] has preliminary implementations of Proactive DSS, based on research done by [18], and Proactive RSA, based on on-going research by Frankel, et. al. The implementations are completely flexible with regard to the number of participants involved in the protocol within the parameters of the particular algorithm. Sandia sees the initial application of proactive protocols in the areas of multilateral international treaties and distributed certification authorities. However, components of the protocols (i.e., secret-sharing mechanisms) have proven valuable to a wide variety of applications. A third implementation is the plan of a DARPA grant project [4].

Acknowledgments

We wish to thank Shai Halevi and Tal Rabin for their comments and contributions in writing earlier versions of this manuscript, and to Hugo Krawczyk for his useful remarks that improved the paper.

References

[1] J. Bar-Ilan, and D. Beaver. Non-Cryptographic Fault-Tolerant Computing in a Constant Number of Rounds. In *Proceedings of the ACM Symposium on Principles of Distributed Computation*, pages 201-209, 1989.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Noncryptographic Fault-Tolerant Distributed Computations. In *Proc.20th Annual Symp. on the Theory of Computing*, pages 1-10. ACM, 1988.

[3] G. R. Blakley. Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conference*, pages 313-317. AFIPS, 1979.

[4] D. Boneh, Ed Felten, Bill Aiello, and Matt Franklin. http://gump.bellcore.com:7700.

[5] D. Carrel and D. Harkins. The resolution of ISAKMP with Oakley, *Internet Draft draft-ietf-ipsec-isakmp-oakley-03.txt*, March 1997.

[6] D. Chaum, C. Crepeau, and I. Damgard. Multiparty Unconditionally Secure Protocols. In *Proc. 20th Annual Symp. on the Theory of Computing*, pages 11-19. ACM, 1988.

[7] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *Proceeding 26th Annual Symposium on the Foundations of Computer Science*, pages 383-395. IEEE, 1985.

[8] R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In Y. Desmedt, editor, *Advances in Cryptology — Crypto '94*, pages 425-438, 1994. Springer-Verlag. Lecture Notes in Computer Science No. 839.

[9] R. Canetti, S. Halevi, and A. Herzberg. Maintaining Authenticated Communication in the Presence of Break-ins. In *Proc. 16th ACM Symp. on Principles of Distributed Computation*. ACM, 1997.

[10] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *Advances in Cryptology — Eurocrypt '97*, pages 103-118, 1997. Springer-Verlag. Lecture Notes in Computer Science No. 1233.

[11] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *Advances in Cryptology — Crypto '89*, pages 307-315, 1989. Springer-Verlag. Lecture Notes in Computer Science No. 435.

[12] W. Diffie, P.C. Van Oorschot and M.J. Weiner. Authentication and authenticated key exchanges. In *Designs, Codes and Cryptography 2*, pages 107-125, 1992.

[13] P. Feldman. A Practical Scheme for Non-Interactive Verifiable Secret Sharing. In *Proc. 28th Annual Symp. on Foundations of Computer Science*, pages 427-437. IEEE, 1987.

[14] Y. Frankel, P. Gemmell, P. Mackenzie, and M. Yung. Optimal resilience proactive public-key cryptosystems. In *Proc. 38th Annual Symp. on Foundations of Computer Science*. IEEE, 1997.

[15] Y. Frankel, P. Gemmell, P. Mackenzie, and M. Yung. Proactive RSA. In B. Kaliski, editor, *Advances in Cryptology — Crypto '97, 1997*. To appear. Springer-Verlag. Lecture Notes in Computer Science.

[16] J. A. Garay. Reaching (and Maintaining) Agreement in the Presence of Mobile Faults. In *Proc. 8th International Workshop on Distributed Algorithms*, Terschelling (NL), September 1994. Lecture Notes in Computer Science No. 857, pages 253-264.

[17] P. Gemmell. An introduction to threshold cryptography. In *CryptoBytes, Winter 97*, pages 7-12, 1997.

[18] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In Ueli Maurer, editor, *Advances in Cryptology — Eurocrypt '96*, pages 354-371, 1996. Springer-Verlag. Lecture Notes in Computer Science No. 1070.

[19] O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game. In *Proc. 19th Annual Symp. on the Theory of Computing*, pages 218-229. ACM, 1987.

[20] V. Hamilton, Sandia National Labs, Personal communication.

[21] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *1997 ACM Conference on Computers and Communication Security*, 1997.

[22] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing, or: How to cope with perpetual leakage. In D. Coppersmith, editor, *Advances in Cryptology — Crypto '95*, pages 339-352, 1995. Springer-Verlag. Lecture Notes in Computer Science No. 963.

[23] IBM Research, the Proactive Security Project. http://www.ibm.net.il/ibm_il/int-lab/Proactive/home.html

[24] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proc. 10th ACM Symp. on Principles of Distributed Computation*, pages 51-59. ACM, 1991.

[25] R. Reischuk. A new solution to the byzantine generals problem. *Information and Control*, pages 23-42, 1985.

[26] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22:612-613, 1979.

[27] A. Yao. Protocols for Secure Computation. In *Proc. 23th Annual Symp. on Foundations of Computer Science*, pages 160-164. IEEE, 1982.

Appendix: Proactive DSS

In order to exemplify an actual proactive signature scheme, we have chosen to outline one such solution, the DSS solution. It is based on the threshold signature scheme of [18] as incorporated with [21]. First, recall the definition of the Digital Signature Standard (DSS), the adopted US standard for signature scheme:

Key Generation: Let p be a prime number of a specified length, q be a prime divisor of $p - 1$ and g be an element of order q in Z_p^*. The triple (p, q, g) is public. The DSS key consists of a secret key x, which is a random number satisfying $1 \le x \le q$, and a verification public key y satisfying $y = g^x \bmod p$.

Signature Algorithm:
Input: m, the hashed message.
Signature: pick a random number k between 1 and q, calculate $k^{-1} \bmod q$ and set

$$r = (g^{k^{-1}} \bmod p) \bmod q \qquad s = k(m + xr) \bmod q$$

Output: the signature on m is the pair (r, s).

Verification Algorithm:
Verify that

$$r = (g^{ms^{-1}} y^{rs^{-1}} \bmod p) \bmod q$$

Now, assume the existence of protocols to perform the following three tasks:

- [JRSS] Joint Random Secret Sharing: servers jointly generate a random value v which is secretly shared among the parties. This task is performed as follows. Each server i chooses a random t-degree polynomial $f_i(X)$. Server i then sends to server j the value $s_{ij} = f_i(j) \bmod p$. Server j then computes its new share $\hat{s}_j$ where $\hat{s}_j = s_{1j} + \ldots + s_{nj} \bmod p$. Observe that the new shares $\hat{s}_i$ lie on the polynomial $\hat{f}(X) = f_1(X) + \ldots + f_n(X)$ which is still of degree t, and its free term $\hat{f}(0)$ is the joint secret v. An extended version of the above protocol which uses verifiable secret sharing also provides, at the end of the protocol, the value $g^v \bmod p$ to all parties.

- [Recp] Given a shared secret v, the protocol for computing the reciprocal computes the shares of its reciprocal $v^{-1} \bmod q$ without revealing v or v^{-1} [1,18]. The value of $g^{v^{-1}} \bmod q$ is also provided to all parties.

- [Mult] Given two shared secretes u and v, compute the shares of the product uv without revealing the original secrets [2,6,18].

Equipped with these tools we are now ready to outline the proactive DSS scheme:

1. Proactive DSS Key Generation: Jointly generate a DSS key by using the [JRSS] protocol to share a random value x. x is now the signing (private) key, and its corresponding verification key is g^x.

2. DSS Signature Generation:

   (i) Jointly generate a shared random value k by using the [JRSS] protocol.

   (ii) Compute the shares of k^{-1} and $r = g^{k^{-1}} \bmod q$; this is done via the protocol [Recp] for computing the reciprocal of the joint secret k.

   (iii) Use the multiplication protocol [Mult] on k and $(m + xr)$ to compute the shares of the value $s = k(m + xr)$. Note that since m and r are known to all parties, and x is a shared secret, the shares of the value $(m + xr)$ are readily known to each server.

   (iv) Each server outputs the pair (r, s_i).

   The shares (r, s_i) are then combined (by a public procedure) to produce the signature (r, s).

3. Proactive Key-Refresh: Use the proactive secret refresh scheme described in the section *Proactive secret sharing* to refresh the key x.

Fast Generation of Random, Strong RSA Primes

Robert D. Silverman
RSA Laboratories
20 Crosby Drive
Bedford, MA 01730 USA

A number of cryptographic standards currently under development place restrictions on the primes that are used in the generation of an RSA key. In particular, in section 4.1.2 of the X9.31-1997 standard for public key cryptography there are a number of recommendations regarding the generation of primes that make up an RSA modulus including that they be "strong."

In this article we will examine these criteria. The position of RSA Laboratories is that virtually all of these requirements are unnecessary [10,11]. In particular, we will show that the relevance of strong primes to the security of RSA is, at best, doubtful. However, given this position, we will outline in this article a fast way of generating random strong primes that also satisfy a number of other cryptographic requirements. The method requires no more time to generate strong primes than it takes to generate random primes.

Are strong primes necessary?

A strong prime p is one for which $p \pm 1$ has at least one large prime factor. The historical reason for this requirement has been to guard against the Pollard $P - 1$ and Williams $P + 1$ factoring algorithms. A paper by Bach and Shallit has shown that algorithms exist for finding a factor p provided that any one of a number of other cyclotomic polynomials in p, e.g. $p^2 + 1$, $p^2 \pm p + 1$, etc. has only small prime factors.

In [10,11] it is argued that the Elliptic Curve Factoring Algorithm (ECM) obsoletes the requirement to protect against these attacks in the sense that generating an RSA key with strong primes does *not* add any security to the key over using random primes. This is because ECM majorizes the $P \pm 1$ algorithms.

There are currently two classes of factoring algorithms that are applied to large integers. The first class attempts to find a factor $p \mid N$ by computing a multiple of a random element in some specially selected group such that the order of that group is divisible only by small primes. The second class includes algorithms such as the Multiple Polynomial Quadratic Sieve (MPQS) and the Number Field Sieve (NFS) and their variations, and will not be discussed here.

The $P - 1$ method uses the group $\mathbb{Z}/p\mathbb{Z}$ in the hope that the order of that group, $p - 1$, will be divisible only by small primes. The $P + 1$ method works in the multiplicative sub-group of order $p + 1$ in the finite field $GF(p^2)$. These two algorithms can be quite effective at quickly finding small prime factors (5 to 25 digits) of large integers. ECM works by selecting a random Elliptic Curve over $\mathbb{Z}/N\mathbb{Z}$ and a random point on the curve. One then hopes to find a multiple of that point which is the identity element mod p, but is not the identity element in $\mathbb{Z}/N\mathbb{Z}$. This will reveal p. This is easy to do if the order of the curve is divisible only by small primes.

The advantage that ECM has over $P \pm 1$ is that if one curve fails it is possible to choose a different curve and hope that its group order is divisible only by small primes. With $P \pm 1$ there is only one group to work with. If that one fails, you are out of luck.

Suppose we choose our primes for our RSA key such that $p \pm 1$, $q \pm 1$ have no small factors and are thus inaccessible to $P \pm 1$. This does not guard against the existence of a small value of k, $k \neq 1$, such that $p \pm k$ is divisible by only small primes. And if such a k exists, ECM can succeed where $P \pm 1$ fails. It is impossible to guard against all such possible values of k.

$P - 1$ has been in use since about 1975. ECM has been in use since 1985. Since then, the largest prime ever found by $P - 1$ was 34-decimal digits (with an FFT version [12]). The largest factor ever found by ECM was 47 decimal digits. This is thought to have been an extraordinary bit of luck. The second largest factor ever found was 43 digits. In the last 12 years, only about a dozen factors greater than 2^{128} have been found. Finding a 50-digit factor should take about 4 times this total effort, and finding a 256-bit factor should take about 50 million times as this total effort.

Suppose one expends the same level of effort with $P - 1$ as was spent in factoring RSA-130 with the number field sieve. It can be shown that a 256-bit prime factor can be found with a probability of 9×10^{-7} and a 384-bit prime factor can be found with

Generating an RSA key with strong primes does not add any security to the key over using random primes.

Robert D. Silverman is Senior Research Scientist at RSA Laboratories East. He can be contacted at bobs@rsa.com.

a probability of 2×10^{-11}. For ECM it is very difficult to estimate the chance of success within an *a priori* fixed amount of time. Extensive computations would have to be performed to determine the optimal ECM parameters for a given level of event when looking for 256-bit and 384-bit primes. A rough estimate suggests 2 to 3 orders of magnitude greater chance than with $P - 1$. But one still only has about a 1 in 10,000 chance of finding a 256-bit prime. In short, randomly chosen 256-bit and 384-bit primes are safe from attacks via $P \pm 1$ and ECM.

In short, randomly chosen 256-bit and 384-bit primes are safe from attacks via P ± 1 and ECM.

X9.31 criteria for RSA key generation

Here we summarize the criteria currently cited in the X9.31 standard for public key cryptography:

> If e, the public exponent, is odd, then e shall be relatively prime to $p - 1$ and $q - 1$.

This is easily satisfied by choosing e.g. $e = 3$, or $e = 2^{16} + 1$. These are commonly used values. This criterion is necessary in order for encryption to work properly. When constructing the primes p and q, it is easy to ensure that e does not divide $LCM(p - 1, q - 1)$.

> If e is even, then it must be relatively prime to $(p - 1)/2$ and $(q - 1)/2$, and $p \not\equiv q \bmod 8$.

These criteria are easily satisfied by letting e be twice a prime, and then generating p and q so that one of them is congruent to 3 mod 8 and the other is congruent to 7 mod 8 with e coprime to $LCM(p - 1, q - 1)$. These latter conditions are easily satisfied during prime generation and we show how to do so below.

Note: The public exponent e is selected prior to generation of the primes.

> The modulus shall have $1024 + 256x$ bits for $x = 0, 1, \ldots$.

As a result, the primes p and q shall then be $512 + 128x$ bits each. The choice of the value of x depends on the level of security required. Larger values of x give greater security. This requirement is a statement reflecting the current state of the art in factoring technology. With the Number Field Sieve 512-bit moduli are simply not secure today for new applications, 768 bits offers a basic level of security, and 1024-bit moduli are often used for longer-term security.

> p and q shall each pass a probabalistic primality test where the probability of error is less than 2^{-100}.

One can also use a deterministic primality *proof* such as the Bosma-Cohen-Lenstra algorithm or the Atkins-Goldwasser-Killian algorithm. The criterion here simply states that we shall have chosen primes with a high degree of confidence; that we have either chosen a prime using a decision procedure and that the probability that the procedure is in error is less than $2^{-100} \sim 8 \times 10^{-31}$ or that we have a rigorous proof of primality.

> $p - 1$, $q - 1$, $p + 1$, and $q + 1$ shall each have large prime factors.

This is the typical strong primes condition. Unfortunately, early drafts of X9.31 do not define 'large'. From the results presented in [5,7,12], we suggest that 2^{100} is sufficiently large. This will put $p \pm 1$ factoring attacks well out of computer range. The size of the prime factors (101 bits) of $p \pm 1$ and $q \pm 1$ is much too large for these algorithms to succeed in the lifetime of the universe.

> $GCD(p - 1, q - 1)$ shall be small.

Early drafts of X9.31 did not define 'small', but the method for generating primes satisfies this requirement sufficiently to guard against the relevant attacks (repeat encryption). The argument in section 9 of [10] shows that if r is a large prime factor dividing $LCM(p - 1, q - 1)$, then either the order of the public exponent exceeds r (which renders repeat encryption attacks impossible because it requires too much computation) *or* the order of the encrypting exponent is small with probability less than $1/(4r)$. Since $p - 1$ and $q - 1$ both have prime factors at least 2^{100}, $GCD(p - 1, q - 1)$ can be no more than $s = \sqrt{N}/2^{100}$, and hence the probability that the public exponent has order less than $k = 2^{100}$ is ks^2/N, which is less than 2^{-100}. In practice, the probability will be much lower. This is a worst case analysis.

> p/q shall not be near the ratio of two small integers and $|p - q| > 2^{412 + 128x}$.

The purpose of these requirements is to guard against Fermat and related (i.e. Lehman) factoring algorithms. This condition is easily checked once one has generated two primes simply by subtracting their 4 or 8 highest order bytes and checking that the difference is non-zero. However, the basis for the requirement is that if p and q are too close together, then Fermat's or Lehman's algorithm can factor the modulus with work load equal to $(p + q)/2 - \lfloor\sqrt{pq}\rfloor$. The algorithm works by finding x, y such that $x^2 - y^2 = N$. The work load cited assumes that the attacker will start with an initial guess for x at $\lfloor\sqrt{N}\rfloor$. However, there is no reason why the attacker can't choose some other starting guess for x, (say) z, and the work then becomes $(p + q)/2 - z$. Requiring $p - q$ to be large can not guard against choices for z other than $\lfloor\sqrt{N}\rfloor$, and hence adds no real security to the key. We view this requirement as irrelevant. Similarly, the requirement that p/q shall not be close to the ratio of small integers is also irrelevant. We do note however, that this last requirement derives from an attempt to guard against the Lehman algorithm. If r and s are small (which in this context means less than 100), then $ps - qr$ needs to be less than 2^{64} for the attack to be feasible, and the chance of this happening is negligible.

$p - q$ shall have a large prime factor.

This condition would prevent an attack that runs in time $N^{1/3}$. In practice this is not a serious threat. Furthermore, the condition seems to be impossible to satisfy, short of actually factoring $p - q$ or running sufficient trials of ECM to be satisfied that no small factor exists.

There is also a suggestion that some forms of the modulus, such as $N = 2^{64x} \pm c$ will simplify the reduction and require less storage. This seems to have been put in place before the discovery of the Number Field Sieve. Moduli of this form are readily susceptible now to the special version of NFS and are quite insecure. They should not be used.

Given that some applications will have to conform to these conditions, we will now describe a method of generating such RSA moduli.

Randomly generating strong primes

We shall randomly generate two strong primes, each of size $512 + 128x$ bits in such a way that their product is $1024 + 256x$ bits. The procedure for prime generation that is outlined below would therefore be executed twice; once for p and once for q. The procedure described here though shall refer to p only. We start by randomly generating a number X of the correct size. Then, we randomly generate 101-bit factors p_1 and p_2 for $p \pm 1$. Using the Chinese Remainder Theorem with p_1 and p_2 we then construct a sequence of candidates, starting at our random point X, for p such that $p_1 \mid p - 1$ and $p_2 \mid p + 1$. We then remove all candidates divisible by small primes with a sieve. Finally, we test the remaining candidates following the sieve for primality.

Selection of starting point

Randomly select an integer X in the range

$$[\sqrt{2}\ 2^{511+128x},\ 2^{512+128x} - 1]$$

Our prime p will be selected as the first integer greater than X which satisfies the strong prime requirements. The product, $N = pq$, of two of these randomly chosen primes will produce the public modulus which will have exactly $1024 + 256x$ bits.

Selection of large prime factors of $p \pm 1$

Start by randomly generating two 101-bit numbers, y_1 and y_2. Using a sieve procedure we shall generate a sequence of candidates for p_1 and p_2 by starting respectively at y_1 and y_2 and sieving out small primes. This will remove a substantial number of composite numbers that need not be tested for primality. We then test what survives the sieve for primality. These shall be p_1 and p_2.

Starting at each of y_1 and y_2 sieve out all small primes up to 10^5 over the range $[y_1, y_1 + 5 \times 10^5]$, and $[y_2, y_2 + 5 \times 10^5]$. The limit, 10^5, for the primes with which we sieve is somewhat arbitrary, and is chosen for reasons of performance, rather than security. Any number between 10^3 and 10^6 is acceptable. Similarly, the length of the sieve interval, 5×10^5, is somewhat arbitrary. One can choose any numbers which are convenient for the particular implementation of this procedure as dictated by resources such as the amount of computer memory available. The length of the sieve interval should be several times the largest prime that is sieved. The numbers selected are a good balance between the cost (in time) of the sieve

There is also a suggestion that some forms of the modulus [...] will simplify the reduction and require less storage. [...] Moduli of this form are readily susceptible now to the special version of NFS and are quite insecure. They should not be used.

procedure against the cost of testing candidates for primality. See the box below for a full description of a sieve procedure.

The length of the sieve interval should be several times the largest prime that is sieved. The numbers selected are a good balance between the cost (in time) of the sieve procedure against the cost of testing candidates for primality.

The sieve will 'strike out' many of the numbers in the sieve interval. The numbers that are removed are divisible by small primes and hence can not be candidates for primality testing. After sieving, test the remaining numbers in the sieve interval sequentially, starting at y_1, y_2 to see if they are prime. Apply 25 iterations of Miller-Rabin to each candidate. This will results in a chance of error of less than 2^{-100}. One can also rigorously prove they are prime (if desired) by applying the Selfridge improvements to the theorem of Proth, Pocklington, & Lehmer [2]. This procedure will yield two primes p_1 and p_2 which will be used as the large prime factors of $p - 1$ and $p + 1$ respectively.

These primes correspond to the $B1$ limit discussed in [10]. It is well beyond computer range to apply the $P \pm 1$ algorithms up to $2^{100} \sim 10^{30}$. In fact, it can't be done within the lifetime of the universe with existing hardware.

Searching for a strong prime

At this point we use the Chinese Remainder Theorem to construct a sequence of integers Y, starting at X such that every integer in the sequence is congruent to 1 mod p_1 and -1 mod p_2. This means that every integer Y in the sequence will have $p_1 \mid Y - 1$ and $p_2 \mid Y + 1$. That is to say, p_1 and p_2 will be large prime factors of $Y - 1$ and $Y + 1$. In order to do this one computes

$$R = ((p_2^{-1}) \bmod p_1) \cdot p_2 - ((p_1^{-1}) \bmod p_2) \cdot p_1.$$

One then computes $Y_0 = X + (R - X \bmod p_1 p_2)$. This is the first integer greater than X which is 1 mod p_1 and -1 mod p_2. Starting at Y_0 one now sieves the integers $Y_0, Y_0 + p_1 p_2, Y_0 + 2p_1 p_2, \ldots$ by all small primes up to (say) 10^6. Once again, the value of 10^6 may be changed to anything that is convenient. The length of the sieve interval should be several times the largest prime in the sieve factor base ($5 \cdot 10^6$ is a good choice). The integers untouched by the sieve will be candidates for primality testing and as a result of our use of the CRT, will automatically be strong primes if they are prime. One also sieves with the public exponent e at this time, so that candidates p with $e \mid p - 1$ are also removed.

A *sieve procedure* is as follows. Start by selecting a *factor base* of all the primes p_i up to some selected limit L. Select a starting point for the sieve P, and a length for the sieve interval M. Compute $S_i = P \bmod p_i$ for all i. Initialize an array of length M to zero. Then starting at $P - S_i + p_i$ let every p_i^{th} element of the array be set to 1. Do this for the entire length of the array and for every i.

Now, every location in the array which has the value 1, is divisible by some small prime and is hence composite.

The array can be a bit array for compactness, when memory is small, or a byte array for speed, when memory is readily available. This is also no need to sieve the entire sieve interval at once before looking for candidations. One can partition the array into suitably small pieces, sieve each piece, look for candidates then go on to the next piece. Every location with the value 0 is a candidate for prime testing.

Even exponents

At the point where one constructs R via the CRT, one could also add in the condition that $R = 3$ mod 8 (for the first prime to be generated) or $R = 7$ mod 8 (for the second prime to be generated) in the case where one wanted to use an *even* integer as the public exponent (the Rabin-Williams system). One can therefore choose Rabin-Williams at essentially no extra computing cost if desired since the additional time computing the CRT is negligible. One also sieves the public exponent as in the odd case.

Testing candidates

Once the set of candidates has been sieved by small primes, one can now test the numbers that have not been touched by the sieve for primality. There are several ways to do this. The set of possible methods has been greatly extended by new results which are discussed below. The basic criteria shall be that any method used must have an error rate no greater then $2^{-100} \sim 7.8 \times 10^{-31}$.

A deterministic primality test

The two best current methods are the Cyclotomic ring test by Bosma-Cohen-Lenstra or the Elliptic Curve Primality Test by Atkin-Goldwasser-Killian [1,8]. My personal recommendation is that while

these are possibilities, they should not be used. The reason is that these algorithms are quite complicated to implement and that the likelihood of error in the implementation far exceeds the likelihood that a random method will return a composite.

Use of the Miller-Rabin algorithm

One applies sufficient tests so that the probability of a randomly generated candidate actually being composite when multiple Miller-Rabin tests say 'prime' is less than 2^{-100}. According to [3,6] for 512-bit primes, 8 iterations suffice. For 640-bit primes, 6 iterations suffice. This is the suggested method for this standard. Other possibilities are given below.

One might use the Miller-Rabin algorithm combined with a Lucas or Frobenius strong probable prime test [4]. According to a very recent paper of Grantham, if one uses a Frobenius probable prime test, the probability that a candidate is composite when the tests say 'prime' is less than 1/1770, as opposed to the 1/4 one gets with Miller-Rabin alone. If one performs a single Miller-Rabin test, followed by T Frobenius tests, the probability of error for 512-bit primes is then less than $1.5 \times 10^{-17}\,(1/1770)^T$ [5, equation 1.6]. To achieve a probability of 2^{-100}, $T = 4$ suffices. One should be able to apply the analytical techniques of [3] to the Grantham algorithm to arrive at even stronger probability estimates. That is to say the number 1.5×10^{-17} can probably be reduced, but the method is as yet too new for anyone to have done this analysis. However, the correctness of the 1.5×10^{-17} bound is not in question.

It should be noted that there is no known composite integer which passes a single Miller-Rabin test, followed by a single Lucas strong probable prime test. Pomerance, Selfridge, and Wagstaff Jr. currently offer $640 for a counter-example. While a formal estimate of the probability of error for a combined Miller-Rabin/Lucas test is still lacking, heuristics suggest that counter-examples are *extremely* rare. This combination of tests was suggested in [9]. It is therefore suggested that following the Miller-Rabin tests a single Lucas test be performed.

This subject area is changing rapidly. The purpose of the above discussions is simply to demonstrate that there are stronger alternatives to Miller-Rabin and they can be used if desired.

Conclusions

Some standards efforts have recommended that restrictions be placed on the form of primes that might be used to produce an RSA modulus. In this article we have outlined reasons why we believe that many of these restrictions are unnecessary; keys generated according to such requirements are, in practice, no more secure than keys that are not. However, given that such restrictions exist in these standards, we have outlined a way that users can generate RSA moduli that will satisfy the conditions.

References

[1] W. Bosma. Primality Proving with Cyclotomy. Doctoral Dissertation Univ. of Amsterdam, 1990.

[2] J. Brillhart, D.H. Lehmer, and J.L. Selfridge. New primality criteria and factorizations of $2^m \pm 1$. Math. Comp. Vol. 29, 1975, 620-647.

[3] I. Damgard, P. Landrock, and C. Pomerance. Average case error estimates for the strong probable prime test. Math. Comp. Vol. 61, 1993, 177-194.

[4] J. Grantham. A Frobenius probable prime test with high confidence. To appear. Available from http://www.math.uga.edu/~grantham/pseudo/pseudo2.ps.

[5] S.H. Kim and C. Pomerance. The probability that a random probable prime is composite. Math. Comp. Vol. 53, 1989, 721-742.

[6] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. Handbook of Applied Cryptography. CRC Press, Boca Raton, 1997.

[7] P.L. Montgomery and R.D. Silverman. An FFT Extension to the $P-1$ Factoring Algorithm. Math. Comp. Vol. 54, 1990, 839-854.

[8] F. Morain. Implementation of the Goldwasser-Killian-Atkin Primality Testing Algorithm. Project ALGO, INRIA, 1988.

[9] C. Pomerance, J.L. Selfridge, and S.S. Wagstaff Jr. The pseudoprimes to 25 … 10^9. Math. Comp. Vol. 35, 1980, 1003-1026.

[10] R.L. Rivest and R.D. Silverman. Are Strong Primes Needed for RSA? To appear.

[11] R.D. Silverman. The Requirement for Strong Primes in RSA. RSA Laboratories Technical Note, May 17, 1997. Available from http://www.rsa.com/rsalabs/html/tech_notes.html.

[12] R.D. Silverman and S.S. Wagstaff Jr. A Practical Analysis of the Elliptic Curve Factoring Algorithm. Math. Comp. Vol. 61, 1993, 445-462.

While a formal estimate of the probability of error for a combined Miller-Rabin/Lucas test is still lacking, heuristics suggest that counter-examples are extremely rare.

DES Challenge Solved

On June 19, 1997 it was announced that the DES challenge, one of the challenges offered as a part of the Secret Key Challenge sponsored by RSA Data Security, was solved. After a search of 96 days and after checking a little under 25% of the key space, the correct key was found revealing the plaintext message "Strong cryptography makes the world a safer place."

While providing a dramatic demonstration of the computing power available across the Internet, the solution of the DES challenge employed nothing more sophisticated than a brute-force search through all possible keys. Nevertheless, since this most basic attack can always be applied to any block cipher and is essentially dependent on the length of the key used, the solution to the DES challenge provides an illustration of the conflicts brought into play when legal restrictions are placed on the length of encryption keys.

After a search of 96 days and after checking a little under 25% of the key space, the correct key was found.

The successful challenge was coordinated by Rocke Verser. During the final 24 hours of the challenge more than 6.4×10^9 DES keys were tried. If the final rate of key testing had been available from the beginning then the DES key would have been recovered in around 32 days. Of course the rate of key testing was increasing as the challenge progressed and so even this time estimate to recover a DES key would have been reduced as more people joined the effort.

A second DES-search effort coordinated from Sweden started much later than the US effort and was beginning to gain ground as it garnered support from sites in most European countries and other countries as far afield as Taiwan. As Rocke Verser was claiming the prize the Swedish effort had searched around 14% of the key space.

Of course there's more to these challenges than mere races. It can be hoped that the experiences gained in these distributed efforts can be used in other endeavors, be it in the search for Mersenne primes (*http://www.mersenne.org/*) or in some distributed effort to factor RSA-type moduli (*http://www.rsa.com/rsalabs/html/factoring/html*).

It is worth noting that the DES challenge was solved using a software-based search. The results would have been particularly interesting if some participant had been tempted to bring hardware techniques into play.

In 1993, a report by Michael Wiener on building a machine dedicated to a similar exhaustive search revealed that for the modest sum of one million U.S. dollars, a machine could be built that would find a DES key in an expected time of three and a half hours. Over the last four years, we can expect that even this remarkable estimate will have improved considerably.

For those interested in more information on the strength offered by different lengths of encryption key, a report on this subject by an ad hoc group of cryptographers and computer scientists was completed in January of 1996. It was published by the Business Software Alliance and is currently available at *http://www.bsa.org/policy/encryption/cryptographers.html*.

And for those interested in further exhaustive search opportunities, there are currently at least three different efforts underway for the solution to the 56-bit challenge using the variable key length block cipher RC5. More information on this and the remaining challenges can be found at *http://www.rsa.com/rsalabs/97challenge/*.

RC2® Published in IETF Forum

Developed in 1987 by Ron Rivest, RC2 is a variable key-length block cipher. It is often used as a drop-in replacement for DES and it features widely in a large number of commercial software packages.

In a move to promote broad acceptance of a single standard for electronic messaging, a description of the RC2 encryption algorithm was recently published for evaluation by the Internet Engineering Task Force (IETF). The published description of RC2 allows developers working in the IETF to scrutinize the algorithm as part of the process to establish S/MIME (which supports the use of RC2) as a standard for electronic messaging.

A description of RC2 can be found at *ftp://ftp.ietf.org/internet-drafts/draft-rivest-rc2desc-00.txt* and for those interested in the progress of the S/MIME standardization effort, more information can be found at *http://www.rsa.com/rsa/S-MIME/*.

Extensive Revisions to PKCS Underway

RSA Laboratories is in the process of revising its well-known PKCS ("Public-Key Cryptography Standards") suite of documents. The PKCS standards have been developed with the assistance of cryptographers and developers from a large number of companies, and the current versions of them give developers guidance and standardization for a variety of cryptographic tasks. The PKCS standards are a key part of important security-conscious protocols such as S/MIME (the dominant protocol for secure email) and SET (the recently-developed protocol for performing on-line credit-card transactions).

As of July, 1997, the PKCS series includes the following documents:

PKCS #1: *RSA Encryption Standard*
PKCS #3: *Diffie-Hellman Key-Agreement Standard*
PKCS #5: *Password-Based Encryption Standard*
PKCS #6: *Extended-Certificate Syntax Standard*
PKCS #7: *Cryptographic Message Syntax Standard*
PKCS #8: *Private-Key Information Syntax Standard*
PKCS #9: *Selected Attribute Types*
PKCS #10: *Certification Request Syntax Standard*
PKCS #11: *Cryptographic Token Interface Standard*

RSA Laboratories recently held workshops in Palo Alto to work on Version 2.0 of PKCS #5 and PKCS #7. Both of these standards are now being modified to be significantly more general and algorithm-independent than their current versions are. It is expected that the upcoming versions of these standards will be used even more ubiquitously than their predecessors are.

With the generous assistance of Chrysalis-ITS and Entrust Technologies, RSA Laboratories also held a 3-day PKCS #11 workshop in Ottawa. Most of this workshop dealt with the content of the upcoming Version 2.01 of the PKCS #11 standard, which specifies an interface for applications to use when they utilize a special-purpose "cryptographic token" to perform cryptographic functions for them. The final Version 2.01 specification should be finished soon.

In addition to the above workshops, RSA Laboratories held a PKCS #12 workshop in Palo Alto to develop the new PKCS #12: *Personal Information Exchange Syntax Standard*. This standard, based on a previous standard written by Brian Beckman of Microsoft, specifies a format for applications to use to transfer cryptographic "personal identity information" such as private keys and certificates from one platform to another. The final PKCS #12 document is keenly anticipated by a variety of key industry players.

Revisions to other members of the PKCS suite are planned for the near future. To keep in touch with latest information on the "next generation" of PKCS, subscribe to the *pkcs-tng@rsa.com* mailing list by sending email to *majordomo@rsa.com* with the line "`subscribe pkcs-tng`" in the message body. There is also a mailing list, *cryptoki@rsa.com*, specifically for following (and contributing to) development of PKCS #11; subscribing is done by sending email to *majordomo@rsa.com* with the line "`subscribe cryptoki`" in the message body.

The PKCS standards are a key part of important security-conscious protocols such as S/MIME and SET.

P1363 Work Continues

Intensive work continues on the IEEE P1363 project, "Standard for Public-Key Cryptography." The standard aims to provide comprehensive treatment of three families of public key techniques: discrete logarithm techniques over finite fields (such as Diffie-Hellman), elliptic curve discrete logarithm techniques (such as ECDSA), and integer factorization techniques (such as RSA). The working group also has plans for a supplement to the standard which will provide treatment of less-established methods.

The project has generated a great amount of interest and continues to receive comments and proposals from members of the cryptographic community worldwide. The working group presented the latest developments to Eurocrypt '97 participants during its meeting that directly followed that conference. The latest editorial contribution was thoroughly reviewed at the working groups' June meeting held in the Chicago area. The next meeting will be held directly following the Crypto '97 conference, and will review a lot of behind-the-scenes work accomplished by the working group members between the meetings. A presentation for the Crypto participants is scheduled for the afternoon of Tuesday, August 19.

The project maintains a mailing list and welcomes comments and participation. Detailed information is available from the working groups' web site, *http://stdsbbs.ieee.org/groups/1363/*.

A N N O U N C E M E N T S

The RSA Data Security Conference '98

The seventh annual RSA Data Security Conference is scheduled to be held in San Francisco on January 13-16, 1998.

Virtually all of San Francisco's Nob Hill will be dedicated to the event, including the Masonic Auditorium, the Fairmont Hotel, the Stanford Court, and the Ritz Carlton.

The conference will deliver four full days of coverage of the latest trends in cryptographic research, product development, market analysis and social thought in the field of cryptography, all presented by some of the leading minds in the industry. An annual pilgrimage for the world's cryptography systems experts, policy-makers, business people and technology developers, the RSA Conference delivers breadth and depth far beyond any other computer security gathering.

Computerworld called it "the *sine-qua-non* event of the crypto community". From very humble beginnings in 1991, when 50 developers gathered to discuss the state of the nascent crypto industry, the annual RSA Conference has grown to become the biggest event on the crypto-circuit. Planners are projecting that over 3,000 cryptographers, policy-makers, business people and technology developers will attend the 1998 conference.

An increasingly significant element of the conference is the RSA Data Security Conference Partner Fair. Scheduled to take place January 13-15 at the Fairmont Hotel, this exhibit hall provides an unparalleled opportunity for attendees to see the very latest developments in cryptographic and computer security products.

For more information or to register, please visit *http://www.rsa.com/conf98/*.

In this issue:

- ***Proactive Security: Long-term Protection Against Break-ins***
- ***Fast Generation of Random, Strong RSA Primes***

For contact and distribution information, see page 2 of this newsletter.

FIRST CLASS
U.S. POSTAGE
PAID
MMS, INC

RSA Laboratories
A Division of RSA Data Security

100 Marine Parkway, Suite 500
Redwood City, CA. 94065-1031

Tel 415/595-7703
Fax 415/595-4126

rsa-labs@rsa.com
http://www.rsa.com/rsalabs

Copyright © 1997 RSA Laboratories, a division of RSA Data Security, Inc., a Security Dynamics Company. All rights reserved.