Responses to NIST's Proposal

The U.S. Government agency NIST has recently proposed a public key digital signature standard [3, 4]. Although the proposal is nominally only "for government use," such a proposal, if adopted, would likely have an effect on commercial cryptography as well.

In this note I review and comment on NIST's proposal. (More correctly, it should be called the NIST/NSA proposal, since the cryptographic algorithm was designed by the U.S. National Security Agency.)

Positive Aspects

The following positive aspects of the proposal are worth noting:

- The U.S. government has finally recognized the utility of public key cryptography.
- The proposal is based on reasonably familiar number-theoretic concepts, and is a variant of the ElGamal [1] and Schnorr [5] schemes.
- Signatures are relatively short (only 320 bits).
- When signing, computation of r can be done before the message m is available, in a "precomputation" step.

Problems with the Proposed DSS

DSS is different from the *de facto* public key standard (RSA). Two-thirds of the U.S. computer industry is already using RSA. Current RSA users include IBM, Microsoft, Digital, Apple, General Electric, Unisys, Novell, Motorola, Lotus, Sun, Northern Telecom, among others. These companies are using industry-developed interoperable standards—the public key cryptography standard (PKCS) [2].

Moreover, DSS is not compatible with existing international standards. International standards organizations such as ISO, CCITT, and SWIFT, as well as other organizations (such as Internet) have accepted RSA as a standard. DSS is not compatible with ISO 9796, the most widely accepted international digital signature standard. Adopting DSS would create a double standard, causing difficulties for U.S. industry that have to maintain both DSS (for domestic or U.S. government use) and RSA (for international use).

DSS also has patent problems. Users of the NIST proposal may be infringing one or more patents. Claus Schnorr claims that DSS infringes his U.S. patent #4,995,082, and Public Key Partners (PKP) asserts that DSS infringes U.S. patents #4,200,770 and #4,218,582. NIST does not give a firm opinion on this matter, and has not made licensing arrangements with either Schnorr or PKP. This leaves potential users of the NIST proposal vulnerable.

To add to the patent confusion, NIST says it has filed for a patent on DSS; a move that has no obvious justification. NIST has not stated why it has filed for a patent. The only motivation I can imagine is that NIST may wish to force users, via licensing requirements, to use key sizes shorter than they might naturally wish to use. (See my discussion of weak cryptography.)

DSS has engineering problems; it's buggy. The verification process can blow up due to division by zero—when $s = 0$ in the computation of t in equation (1). NIST was apparently unaware of the problem, or how to fix it—no mention of this problem is made in the description of DSS.

A simple fix is available: merely recompute the signature (starting with a new randomly chosen k value) whenever it is determined that $s = 0$ in the signing process.

This correction is actually necessary for security reasons as well: a user who produces a signature with $s = 0$ is inadvertently revealing his or her secret key x via the relationship in this case: $x = -H(m)/r \pmod{q}$.

While this bug is easily fixed, its presence certainly makes one wonder about the "quality control" used in the development of the standard.

In addition, DSS is too slow, especially for verification. The signing speed is comparable to (but slightly slower than) RSA, while the verification speed is *over 100 times slower* than RSA. Here, we assume RSA uses a small public exponent such as 3, as is common practice.

This slow verification time is likely to introduce unacceptable performance delays in many applications. I note that almost all public key applications are based on the use of certificates to authenticate public keys. Often, a chain of several certificates must be verified to authenticate a single public key, before it is used. In a typical application then, the ratio of verifications performed to signatures performed is fairly large. In other applications, such as software virus detection, the ratio can be enormous. The claim by NIST that signing speed is more important than verification speed just does not hold for most applications.

NIST's proposal is incomplete in three respects:

- it lacks any mechanism for privacy,
- it does not include a hash function proposal, and
- no mechanism for public key certificates has been proposed.

First, NIST's proposal is only for a public key signature system; there are no specifications for privacy (encryption) or the exchange of secret keys. Many applications require both privacy and authentication (i.e., both encryption and digital signatures). For example, any electronic mail system that aspires to replace paper mail should, at a minimum, aim to achieve what one can get with envelopes and hand-written signatures. It is easy to imagine many other applications in which protection from eavesdroppers may be considered important. For example, home-banking is such an application. So is the transmission of sensitive corporate documents (financial or engineering) from one corporate office to another.

Therefore, DSS is, at best, half a standard—it does not provide the basic functionalities one expects of a full public key system. While NIST has made comments about introducing an encryption capability at some point in the future, one can hardly expect it to happen soon (given NSA's involvement in this standards process).

Second, a *hash function H* was not specified in the original proposal, although one is clearly necessary. (At the time of my writing this article, a proposal for hash function had just been announced; I have not had time to review this proposal.)

Third, NIST has not specified how one might create public key certificates which are necessary in almost all applications of public key cryptography. In essence, public key cryptography depends on the use of authenticated public key certificates. A signature scheme that does not address how certificates are to be created (as does the PKCS standard [2]) is not usable.

DSS Specifications

The following parameters are *global*, and can be shared by many users:

- a 512-bit prime p,
- a 160-bit prime q dividing evenly into $p - 1$,
- an element g of $\mathbf{Z}_p^*$, whose multiplicative order is q,
- a *hash function H* mapping messages into 160-bit values.

The following parameters are *per user:*

- a *secret key x*, where $0 < x < q$.
- a *public key y*, where $y = g^x \pmod{p}$.

So the secret x is the *discrete logarithm* of y, modulo p, with base g.

To sign a message m, a user produces his or her signature as (r,s), by selecting a random value k from $\mathbf{Z}_q^*$, and then computing

$$r = (g^k \pmod{p}) \pmod{q}$$

$$s = k^{-1}(H(m) + xr) \pmod{q}$$

To verify a purported signature (r,s) for the message m, one first computes

$$t = s^{-1} \pmod{q}$$

and then accepts the signature as valid if the following equation holds:

$$r = (g^{H(m)t} y^{rt} \pmod{p}) \pmod{q}.$$

DSS has Security Problems

The bulk of this analysis focuses on the issue of the key size of the proposed DSS.

Shared global moduli are risky. While the NIST proposal does not require users to use a common modulus p, it encourages such a practice. This is a security weakness, since "breaking" that one modulus can compromise the security of all users simultaneously. The current state of computing discrete logarithms includes algorithms that break a modulus p by doing a precomputation that later permits very easy computation of the discrete logarithm of any y value modulo p. Therefore, users who share a modulus are putting all their eggs in the same basket. The NIST proposal should warn users away from such a practice—each user should choose his or her own modulus p.

There is a risk of "trap-doors" in the primes. Arjen Lenstra and Stuart Haber (in their letter to NIST) observed that for some primes the discrete logarithm problem is "easy" for the person who created the prime (thus, a "trap-door prime"). Moreover, it may be difficult for a user to recognize trap-door primes. The question as to how one might be able to effectively create and exploit such trap-door primes is an active area of research; the dust is still being created here, instead of settling down. Thus, conservative users of the proposed DSS should assume that whoever creates the prime p can determine their secret keys x from their public keys y. Again, NIST should have warned users not to use primes created by others—each user should choose his or her own modulus.

The scheme is new and untested. The preceding discussion is based on the assumption that the cryptanalytic problem is the "discrete logarithm problem." In fact, the problem here is a relatively new variation of the classic discrete logarithm problem. The reason is that g has only order q instead of order $p - 1$. It is quite possible that this new problem—breaking DSS—is far easier than the general discrete logarithm problem. Without further research into this new variation, it is difficult to assess the true level of (in)security provided by DSS. Cryptographic proposals need to withstand many years of careful scrutiny before they become plausible candidates for field use. This is especially when the proposal is based on novel mathematical problems.

There is a security weakness if k is disclosed. Signing with DSS requires the generation and use of random numbers (the k values). It is true, but not noted in NIST's proposal, that a user's secret key can be totally compromised if any one of the k values used in creating a signature is ever compromised. The secure creation and utilization of the k values is thus of critical importance to the security of the entire scheme. Use of pseudorandom number generation schemes to generate k involves the risk that the pseudorandom number scheme could be broken. While the NIST proposal makes a few suggestions in this regard, it does not require any particular approach, thus permitting totally insecure choices by the user. (And to make matters worse, the one approach NIST does suggest involves the data encryption standard (DES), which would be unusable in any application involving export.) The NIST proposal is thus fatally incomplete in specifying how to implement the choice of k; the poor user is given enough rope with which to hang himself—something a standard should not do.

Key Size Too Short

I believe that a national standard based on fixed 512-bit key size would serve our country very poorly—such a proposal unnecessarily risks catastrophic failure of the integrity of our financial, industrial, and governmental information-processing systems.

To begin with, I question the rationale for having a fixed key-size at all as part of the proposal. Clearly, one needs a *minimum* key size to prevent users from choosing key sizes that are too short to be secure. And one might want a *maximum* key size in order to design efficient yet fully compatible implementations. Yet there is no obvious reason why the minimum required key size and the maximum allowed key size should be the same.

Indeed, the NIST "one-size-fits all" proposal is a very poor match to the engineering and security needs of most public key applications. Typically, there are many users, each of whom has a public key used by others to verify the user's digital signatures. Such applications are invariably based on the use of *certificates*, so verifiers can assure themselves that they are verifying signatures with the appropriate public key. A certificate is a message associating a user's name with his or her public key; this message is itself signed by a "certifying authority" whose public key is widely known. A typical signature verification then normally consists of *two* verifications: one to verify the certifying authority's signature on the user's certificate, and then another to actually verify the user's signature.

A certificate-based application thus incorporates two basic kinds of signatures: ordinary user signatures and signatures by a certifying authority. The latter kind form the backbone of the integrity of the entire application, since the ability to forge certificates would give an attacker essentially unlimited power to corrupt the system. I consider it essential in secure public key system design to have certifying authorities use the maximum allowable key size. This size is typically much larger than what an ordinary user might select for his or her own public key size. As an example, in a typical RSA-based application today, certifying authorities might use keys of 1,024 bits or more, whereas ordinary users might choose key sizes of 500 to 800 bits.

In a typical application, the trade-off between security and performance mandates the use of different key sizes in different parts of the system. Certifying authorities, or users with very valuable data, must use very long keys to achieve the highest possible security level. Other users, with reduced security requirements and/or more stringent performance requirements, will use shorter keys. Trying to make one-size-fit-all results either in unacceptably low security for all users (because all certificates will be suspect) or unacceptably poor performance for some users.

In a public key system based on number theory, there is no valid technical reason for requiring a fixed key size. The underlying number-theoretic algorithms can support arbitrary key sizes. Users and certifying authorities should be able to choose key sizes according to their requirements.

I now turn to a discussion of the particular key size

NIST has chosen: 512 bits. (By key size, here, I refer to the size of the prime modulus *p*.) I argue here that if one is going to insist on having a fixed key size, then 512 bits is far too short.

I note that NIST provides no justification for its choice of key size.

To estimate the necessary key size, one must understand the computational resources available to an imagined potential attacker, and the computational difficulty of the underlying cryptanalytic problem. Let me address each of these issues in turn.

How much computational power can an imagined attacker bring to bear to break the system? This depends on the time period we are talking about (since technology is rapidly evolving) and the financial resources of the attacker (to purchase the necessary computing power).

It is necessary to know the expected lifetime of the proposed standard in order to know for what level of security to aim. A scheme that is considered "secure" today may not be secure in the year 2000, and a scheme considered secure in the year 2000 may not be secure in the year 2010. Computer technology is evolving at an incredible pace, and is likely to continue to do so for the next few decades. The security of cryptographic schemes thus tends to erode steadily over time, and the design of cryptographic systems must envision and plan for such erosion.

I suggest that a digital signature standard should be designed with a minimum expected lifetime of at least 25 years. That is, one should design so that a system adopted in the year 1992 should still be secure in the year 2017. It should not be possible for an attacker in 2017 to forge a signature, using the computers available then.

Where does "25 years" come from? To consider the only available precedent of the lifetime of a NIST cryptographic standard, I note that the DES was adopted in 1976 and seems likely to be in widespread use by 1996. After a cryptographic signature standard has been terminated, an additional period of time is required during which the validity of signatures can still be assured. For example, it is not uncommon to require that signed documents be retained and be considered legally binding for seven years. A signature produced in the year 2010 should still be verifiable in the year 2017, with an understood assurance that it was not just recently forged. I consider a 25-year expected lifetime a minimum reasonable requirement for a digital signature standard.

What kind of computational power will be available to an attacker in the year 2017? It is widely asserted that computational power (per dollar spent) is increasing at approximately 40% per year. This means we have an approximate doubling of computer power (per dollar) every two years, and an approximate increase of a factor of 4,500 after 25 years. Let's round this off to 5,000 for our back-of-the-envelope calculations (corresponding to 25.3 years at 40% growth/year). In the year 2017, I expect computer power will be about 5,000 times cheaper than it is now.

How big an attack should one prepare for? Let me suggest that a national digital signature standard should, at a minimum, be able to withstand an attack costing an attacker $25 million (in today's dollars). This amount of money is easily available to large corporations, drug dealers, and small countries. There is no reason that our national security, in terms of the integrity of our electronic business, financial, and governmental information-processing systems, should be vulnerable to an attack costing only $25 million. Indeed, it is easy to make an argument for a much higher threshold; it is not hard to imagine scenarios in which the benefit of a successful attack exceeds $25 million. However, I'll continue our back-of-the-envelope calculation with the $25 million figure.

How much computing power can one buy for $25 million? Today, a workstation with 100MIPS can probably be purchased in quantity for about $5,000. An attacker wouldn't need all of the peripherals (screen, floppy disk, mouse, nice cabinet), and could economize by sharing such equipment as power supplies, and fans. He or she is basically interested in having many processors, each with a reasonable amount of memory. Let me estimate that such a "stripped-down" 100MIPS processor would have an amortized cost today of $1,000.

A convenient unit of computational work is a MIPS-year—the amount of computation performed by a 1MIPS processor running for a year. A MIPS-year thus corresponds to about 32 trillion basic operations. If we assume that a 100MIPS processor lasts for about 10 years, we obtain an amortized cost estimate for today of $1 per MIPS-year of computation. (Here we are buying "computation by the yard"; our yard is one MIPS-year, and it should cost about $1 in quantity. I leave the details of buying computational power in 2017 to your imagination; a simple cost-effective way might be to spend considerably more than $25 million to purchase hardware, and then to resell the hardware after the computation is done.)

This analysis is a bit naive, and the straight-line depreciation over 10 years is rather inappropriate to use when the underlying technology is changing so quickly. With the cost of computation decreasing by a factor of 1.4 every year, the value of the computational power of some purchased hardware should be $(1 - 1/1.4) =$ 29% in the first year and the remaining 71% in the remaining years. Thus the cost per MIPS-year during the first year would be about 0.29 $1000/100 = $2.90 per MIPS-year. If one calculates in a 10%-rate of return on funds as well, the cost per MIPS-year rises to $3.57 (details omitted). Taking these considerations into account, and recognizing that there are other unaccounted-for factors (ex., power supply, maintenance), we can estimate that the cost per MIPS-year today is about $4.

We therefore can estimate that an attacker with $25 million to spend today could purchase about 6.25 million MIPS-years of computation. In the year 2017, the attacker will be able to purchase about 5,000 times as much, or 31.25 billion MIPS-years. I believe that a digital signature standard adopted today should, at a minimum, be able to withstand an attack of 31.25 billion MIPS-years. (This may sound like a lot of computation, but you can see from my arguments that this is, in fact, a rather conservative estimate of the security requirement for such a standard.)

How large a key size is needed to withstand an attack of 31.25 billion MIPS-years? This depends, of course, on the cryptanalytic problem to be solved. In the case of the proposed DSA, the basic cryptanalytic problem is assumed to be the classic discrete logarithm problem: computing x, given g, p and $g^x \bmod p$. Using the best-known algorithms for this problem, the number of operations required is approximately

$$L(p) = e^{\sqrt{\ln p \ln \ln p}}.$$

The state of the art in algorithms for the discrete logarithm problem is still evolving, but this formula certainly seems like a very conservative estimate of what will be possible in 2017, since it represents what is possible today. For example, with a 512-bit prime p (as in NIST's proposal), we see that only

$$L(2^{512}) = 6.7 \times 10^{19} \text{ operations} \qquad (2)$$

$$= 2.1 \text{ million MIPS-years} \qquad (3)$$

of computation are required to break a 512-bit problem. Thus, we see that *the proposed DSA is over 14,000 times weaker than a conservative analysis suggests is required.*

Another way of stating this result is that *the DSA, as proposed, is breakable today within the resource bounds suggested as reasonable.* (Indeed, purchasing 2.1 million MIPS-years today should cost only $8.2 million, far less than the $25 million suggested as reasonable for an attacker to have available.)

Setting $L(p)$ equal to 31.25 billion MIPS-years, and solving for p, we find that *the DSA should be using keys of at least 710 bits,* minimum.

As noted, this is a conservative estimate. It doesn't plan for improvements in the state of the art of algorithms for solving the discrete logarithm problem, which can have a dramatic effect on the key size required. (Indeed, there are already algorithms "waiting in the wings" that are theoretically faster than this analysis envisions, which may turn out to be faster in practice as well.) The estimate has no margin built in for faster-than-expected improvements in hardware—some of my colleagues suggest that doubling every 18 months (a cost/performance improvement rate of 59% per year) is more realistic than doubling every two years; such a rate gives the adversary another factor of 23 in capability over 25 years. The analysis also does not plan for longer-than-expected adversaries. The ability to harness for free "unused" processor cycles over large networks of workstations could also dramatically increase the computational ability of an adversary, thereby altering the equation further in his favor.

The previous discussion focuses on the choice of p. A similar discussion applies to the choice of q. The security of the scheme certainly depends on the choice of q as well as the choice of p. There is reason to wonder whether a fixed 160-bit size for q is sufficiently secure in the face of rapid technological advance. Again, it would be best if users could adjust the size of q they select (within certain generous limits) to take into consideration such uncertainties.

For these reasons, and as a matter of sound conservative design, I believe that a substantial margin of safety should be built into the standard. Most important, certifying authorities should have generous key sizes allowed. *I would strongly recommend that any digital signature standard based on the discrete logarithm problem should allow certifying authorities to use key sizes of 1,100 bits or larger. Similarly, ordinary users should be allowed key sizes of at least 850 bits.* Finally, users should be allowed greater freedom in the selection of the size of their q values. I feel that anything less is short-sighted and risky, possibly verging on the irresponsible. In cryptographic systems, responsible design is conservative design; generous allowance must be made for unforeseen developments.

The Selection Process Is Flawed

The process by which the DSS proposal was generated, and by which it is being evaluated, seems to be flawed.

It is curious that NIST did not publish a Call for Proposals for a signature algorithm. I note that NIST did so when the DES was adopted. (The DES arose from a proposal from IBM.) DSS was created without any such public solicitation for proposals. This is especially curious given the much greater cryptographic expertise now available outside the government, and the much larger base of public key applications already fielded. The DSS algorithm was created by the NSA, and adopted by NIST as its proposal, without any input from U.S. industry.

The closed-door approach toward the development of DSS, together with NIST's assertion regarding patents (see the prior discussion) have created a confrontational, rather than a cooperative, situation between NIST and the U.S. industry.

The very short comment period has not helped. The initial comment period (three months) was extended by NIST to six months. Even so, such a comment period is not sufficient to perform the mathematical study required to validate a new proposal. I note that there are two major worldwide technical conferences on cryptography each year (CRYPTO and EUROCRYPT); the NIST comment period doesn't include either conference. For some reason NIST has not even provided the potential user community a chance to gather and discuss the proposal at its usual cryptographic conferences, the natural place for the technical merits of such a proposal to be debated.

Furthermore, NIST has actively hindered the evaluation process by keeping secret all information about what criteria were employed in the design and selection of DSS. A Freedom of Information Act (FOIA) request by Computer Professionals for Social Responsibility (CPSR) for this information has been denied by NIST (this denial may be illegal, and is being appealed).

Speculation and Discussion

Why is the NIST proposal what it is?

It is my belief that the NIST proposal represents an attempt to install weak cryptography as a national standard, and that NIST is doing so in order to please the NSA and federal law enforcement agencies. The NSA may prefer that cryptographic expertise and capability outside its own walls remain at a minimum tolerable level, to maximize the effectiveness of its foreign intelligence operations. A U.S. standard, even if weak and flawed, may be widely used overseas, making NSA's job easier. Federal law enforcement agencies may fear that good cryptography may render ineffective its traditional use of court-ordered wiretaps. While the DSS is nominally a proposal for only a signature standard, there are several public key encryption algorithms known that could make use of distributed DSS public keys. A strong signature algorithm invites extension to a strong public key encryption algorithm; concern about this possibility is probably the major reason NIST selected a scheme based on "weak cryptography" as its proposal. Should DSS be extended later to a public key encryption standard, weak cryptography will then be built into the national encryption standard, as well as the national signature standard.

The U.S. has the largest information-based economy in the world; we thus have the most to lose by installing weak cryptography as a national standard. The risk is greatly magnified if the standard is not only weak, but *brittle* because users are sharing the same modulus.

Because of rapid technological change, a standard based on weak cryptography will have to be replaced or upgraded all too soon. This is happeing today with the DES, which was arguably a "weak cryptography" standard. DES is clearly near the end of its lifetime. The cost of replacing DES in all of the applications that use DES will be very large. The cost of having to do so in a few years for a weak signature standard will be enormous.

Weak cryptography implements the wrong social policy. It empowers those with large amounts of computing power, and only those, to break in. There is an implicit assumption here that only the government could have enough computing power to break in, thus court-ordered wiretaps would remain effective, even if cryptography is used. But in the face of rapid technological change and a national budget crisis, this assumption is dubious.

Are there technical alternatives that would satisfy all parties? Perhaps. It is certainly the case that the formulation of the problem to be solved has never been made explicit for the cryptographic community to work on. I suspect that a solution based on "escrowed secret keys" might be workable, wherein each user is legally required to deposit his or her secret key with a trusted third party, such as the user's bank. Cryptographic hardware and software would only operate with public keys that were certified to having their corresponding secret keys appropriately escrowed. A federal agency could then obtain the secret key, or its use, with an appropriate warrant. Once their secret keys were escrowed, multinational corporations could even operate across borders with a high degree of authentication and privacy (except from court-ordered wiretaps). Cryptographic hardware and software manufactured in the U.S. would not operate abroad without public keys suitably certified as having their secret counterparts escrowed in the U.S. In an extension of this approach, users can escrow their secret keys with several trusted third parties in a "secret-sharing" manner, so that no single third party can compromise the user's key. While this approach may have its own difficulties, it does illustrate that weak cryptography is not the only technical approach available. There may be much better techniques for achieving a compromise between a number of conflicting national concerns.

I believe that the NIST proposal represents an attempt to achieve a compromise of these concerns without public discussion or debate. Unfortunately, the participants in the private discussion were mostly on one side of the table. Perhaps a more public review of the issues that really motivated NIST's proposal will help to form a national consensus on how to proceed.

Conclusions

For the reasons stated here, I feel the proposed DSS, with its fixed (512-bit) key size, is not sufficiently secure to be acceptable as a national signature standard. Indeed, the only purpose to be served by requiring a fixed key size is the installation of weak cryptography as a national standard. Instead, users should be free to choose their own security level by selecting their own moduli and key lengths, up to some very generous maximum limit.

Furthermore, the process by which DSS was selected is seriously flawed. To rectify matters, I believe NIST should:

• Withdraw DSS as a proposal, based on the overwhelming negative response DSS has received during its "request for comments" period,
• Initiate a frank public discussion of our national cryptographic objectives, followed by a reconsideration of what standards are needed to achieve those objectives. As part of NIST's contribution to this discussion, NIST should respond positively to CPSR's FOIA re-

quest for information regarding how DSS was selected,
• Lead explicit public discussion of whether weak cryptography should be a basis for a national cryptographic standard. Users of such a standard should know whether it was chosen because it is breakable in practice,
• Rescind its memorandum of understanding with NSA, whereby NSA was essentially given effective control of the standards process through NIST,
• Restart the process of selecting a national public key standard by soliciting proposals from industry and academia for such a standard.

It is past the time for national cryptographic standards to be designed in secret backroom negotiations according to hidden agendas. NIST should assume a leadership role by abandoning its current proposal and starting fresh.

Ronald L. Rivest
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Mass.

Ronald L. Rivest, along with Adi Shamir and Leonard Adleman, invented the RSA algorithm in 1978.

References

1. ElGamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory 31* (1985), 469–472.
2. Kaliski, B.S. Jr. An overview of the PKCS standards. Tech. Rep., RSA Data Security, Inc., June 1991.
3. National Institute for Standards and Technology. Digital signature standard (DSS). *Federal Register 56* (Aug. 30 1991), 169.
4. National Institute for Standards and Technology. A proposed federal information processing standard for digital signature standard (DSS). Tech. Rep. FIPS PUB XX, National Institute for Standards and Technology, Aug. 1991. DRAFT.
5. Schnorr, C.P. Efficient identification and signatures for smart cards. In G. Brassard, Ed., *Proceedings CRYPTO 89,* Springer, 1990. Lecture Notes in Computer Science No. 435. pp. 239–252.

I am responding to your request for comments on the proposed DSS. My detailed comments follow, but I can summarize by saying that I am deeply concerned by faults in the technical specifications of the proposed DSS and its development process.

NIST has lost considerable credibility with the non-military cryptographic research community and, unless the revision process of DSS is carried out in a much more rapid and open fashion, NIST is likely to become totally ineffective in the setting of cryptographic standards. Since that would be a grave loss to both NIST and the nation, I hope change is possible.

I look forward to seeing your response to these concerns.

1. DSS does not include key exchange. Public key cryptography provides two advantages over conventional cryptography:

Key exchange: the ability for users to communicate privately without fear of being overheard, and without using couriers, registered mail, or similar means for prearrangement of a secret key

Digital signatures: the ability to sign messages which are easily checked by anyone, yet which cannot be forged or modified, even by the intended recipient.

The DSS addresses only the second of these two needs. Until a key exchange standard is developed, users who follow the standard will be at a severe disadvantage in terms of the privacy of their communications. It would have been a simple matter for NIST to include a key exchange standard with the DSS, either by adopting the RSA system [5] for both operations or by specifying the Diffie-Hellman key exchange system [1] as the standard to be used with the current DSS. (The Diffie-Hellman system is the natural key exchange choice if the proposed DSS is used for digital signatures. The DSS is derived from the Diffie-Hellman system.) Because of its early publication (1976), this system possesses a high degree of confidence regarding its security level, as discussed under #4.

2. The key size is too short. The proposed DSS is restricted to a 512-bit modulus or key size. It is generally accepted in the cryptographic research community that this is too short for a system such as DSS, which is based on discrete logarithms and which requires a long life. To quote from a recent paper written prior to the announcement of DSS [3], "even 512-bit primes [key size] appear to offer only marginal security." This is such a well-established viewpoint that further explanation seems unnecessary. While there should be lower limits on the key size to ensure a reasonable level of security, there is no reason for an upper limit. If, in spite of this argument, NIST keeps an upper limit, it should be increased to *at least* 1,024 bits.

The proposed DSS also limits the "subkey" size to 160 bits, a value that is again too short.[1] It is possible to

Rather than claim an advantage for DSS or RSA in a particular environment, I believe the best standard is one which is usable in the greatest number of environs.

recover a user's secret key x from his or her public key y in 2^{80} operations by using 2^{80} words of memory [4]. (Any other breakdown can be used as long as the time memory product is 2^{160}—for example 2^{100} operations and 2^{60} words of memory.) While such a computation is currently infeasible, its possibility is closer than seems comfortable, considering probable advances in technology and improvements in algorithms. The particular cryptanalytic problem involved in breaking the DSS has not been well studied (see #4), making such improvements highly probable. As with the 512-bit modulus, the subkey length should not have an upper bound. Or, if NIST insists on keeping an upper bound, it needs to be at least double, and preferably, quadruple the current 160-bit value.

3. NIST does not provide adequate warning on the danger of using DSS as a common modulus system. While common modulus systems have an advantage in speed of key generation, they allow a successful attack on one user's secret key to be extended to all users of the common modulus. Using a common modulus is analogous to having all personnel within an organization use combination locks with 10-digit combinations, but with the first nine digits being common to all users. This simplifies setting the combination of a lock, but allows an opponent to amortize the cost of an attack on one lock over the large number of locks that are then easily picked.

While DSS need not be used in common modulus mode and there are some applications where that mode is desirable, clear warnings are needed about reduced security in common modulus mode. The proposed DSS says the modulus "can be common to a group of users" without any mention of the attendant danger.

Use of a common modulus would be of less concern if the key and subkey sizes of DSS were increased as suggested earlier.

4. DSS is based on a system which has had limited time for appraisal. Cryptography is still more an art than a science. For most systems, including all digital signature systems, proofs of security are currently impossible. Instead, we rely on concerted attacks by "friendly opponents" intent on fame, rather than thievery, if they are successful in breaking the system. We become more confident of the security of a system as it is subjected to widespread public scrutiny for long periods of time.

The DSS is based on Schnorr's variant [6] of the El-Gamal signature scheme [2]. Thus, there has been little time to gain confidence in Schnorr's variation. El-Gamal's system, while older, is still only half the age of its primary competition as a digital signature standard, the RSA system [5].

Unless Schnorr's scheme possesses some major advantage compared to RSA, it is strange that Schnorr's scheme was selected as the standard. I am aware of no such major advantage of Schnorr over RSA. The only advantage I see is that Schnorr's signatures are somewhat shorter (320 bits versus 512 bits for a comparable security RSA using today's best-known algorithms). On the other hand, in addition to possessing a higher confidence level regarding its security, RSA has a major advantage over Schnorr: Using RSA would automatically have provided for public key exchange, a critical part of the public key standard that NIST has not yet developed (see #1).

5. NIST ignored the danger of probable improvements in cryptanalytic algorithms. Cryptanalyzing the DSS is a special case of computing a discrete logarithm. The history of this problem, as well as the closely related problem of factoring, shows a slow but steady improvement. It was only about 15 years ago that the subexponential nature of the problem was realized. Prior to that time, estimates of the effort required to break DSS with a 128-bit key would have been beyond the realm of reasonability, while today, even a 256-bit key would be insecure.

Improvements over the last 15 years in finding discrete logarithms have effectively cut key sizes by a factor of four. Should that happen again over the next 15 years, the DSS would be totally insecure. For similar reasons, I have always advocated at least a factor of two, and preferably a factor of four, as a safety margin. The proposed DSS imprudently has little or no safety margin.

The danger is increased because of recent advances. Until two years ago, all subexponential algorithms for discrete logarithms and for factoring took time of the form $\exp[k \ln(n)^{1/2} (\ln\ln(n))^{1/2}]$ with $k = 1$ as the best value. Recently, number field sieves have been proposed that solve both problems in time $\exp[k \ln^{1/3}(n) (\ln\ln(n))^{2/3}]$ with k approximately equal to 2. While the higher value of k makes the new algorithm no better for 512-bit keys (1E20 operations versus 7E19 operations for the earlier algorithms), it is probable that the value of k will be reduced as attention becomes focused on number field sieves.

Over the last 15 years, algorithms requiring $\exp[k \sqrt{In(n)\, InIn(n)}]$ operations have been improved from $k = 2$ to $k = 1$. It would be prudent to assume similar advances in number field sieves, in which case breaking DSS would become trivial, requiring only 1.0E10 operations, a computation that can be done on a personal computer.

6. NIST has misstated patent licensing requirements. Raymond G. Kammer, Deputy Director of NIST, has stated that "the digital signature standard is expected to be available on a royalty-free basis in the public interest worldwide.[2] Yet at least two privately owned U.S. patents cover the DSS (#4,200,770 and #4,218,582).

I understand that Schnorr is claiming that his patent (#4,995,082) is also needed to practice the DSS. If Schnorr's claim holds, then the DSS has a patent disadvantage compared to either RSA or ElGamal/Diffie-Hellman since the U.S. government has the right to use the latter systems on a royalty-free basis, but I doubt that it has rights to Schnorr's work. (Clarification from NIST would be appreciated.)

7. NIST has confused the issue of speed comparisons. In his previously referenced statement, Kammer also stated that

> . . . the digital signature technique [DSS] provides for a less computationally intense signing function than verification function. This matches up well with anticipated federal uses of the standard. The signing function is expected to be performed in a relatively computationally modest environment such as with smart cards. The verification process, however, is expected to be implemented in a computationally rich environment such as on mainframe systems or super-minicomputers.

Under the environment specified by Kammer, the DSS would have an advantage over RSA, the primary competing technique. As a universal standard, however, the DSS will often be used in complementary environments where signing is done in a computationally intense environment and verification in a computationally modest environment. A good example is the use of digital signatures generated by a bank and checked by a customer on his or her home computer. This environment would favor "small exponent" RSA systems which allow verification to be performed with approximately 1% of the total signing effort of DSS (including precomputation).

Rather than claim an advantage for DSS or RSA in a particular environment, I believe the best standard is the one which is usable in the greatest number of environments envisioned for its use. That approach leads to the most widely applicable standard. On that basis, ElGamal or Schnorr's signature scheme is approximately equal to RSA. Hence, none of the competing systems should be deemed to have a speed advantage over the others.

8. The adoption process appears to have been conducted in secret. Although listed last, this is the most important change since a more open adoption process would have avoided most of the DSS shortcomings listed here.

I am not aware of any attempts on NIST's part to involve researchers in academia and industry. (If there were such attempts, I hope NIST will make these a matter of public record.) Rather, NIST appears to have worked in secret with only NSA providing advice. This is dangerous because much of NSA's legally mandated mission involves foreign espionage which would be hampered by secure public encryption. As with any organization or individual, NSA is likely to put greater emphasis on its concerns than would a neutral third party.

There is an unavoidable trade-off in that providing a high level of communications security to American business and citizens also makes this protection available to our foreign adversaries. NIST's actions give strong indications of favoring protection of NSA's espionage mission at the expense of American business and individual privacy. While an impartial working group might conclude that such a policy was in the nation's best interests, relying solely on NSA for advice is unlikely to produce an optimal trade-off for the nation as a whole.

Martin E. Hellman
Professor of Electrical Engineering
Stanford University
Stanford, CA 94305

Martin E. Hellman, along with Whitfield Diffie and Ralph Merkle invented public key cryptography in 1976.

References

1. Diffie W., and Hellman, M.E. New directions in cryptography. *IEEE Trans. Inf. Theory IT-22* (1976), 472–492.
2. ElGamal, T. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory IT-31* (1985), 469–472.
3. LaMacchia, B.A., and Odlyzko, A.M. Computation of discrete logarithms in prime fields. *Design, Codes, and Cryptography,* vol. 1, 1991, pp. 47–62.
4. Pohlig, S.C., and Hellman, M.E. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Trans. Inf. Theory IT-24* (1978), 106–110.
5. Rivest, R.L., Shamir, A., Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21* (1978), 120–126.
6. Schnorr, C.P. Efficient identification and signatures for smart cards. *Advances in Cryptology: Proceedings of Crypto '89,* G. Brassard Ed., *Lecture Notes in Computer Science* 435, Springer-Verlag, N.Y., pp. 239–251.

[1] I am indebted to Leonard Adleman of USC for pointing this out.

[2] June 27, 1991 statement before the Subcommittee on Technology and Competitiveness of the Committee on Science, Space and Technology of the House of Representatives

In response to the proposed DSS, we would like to offer a few constructive comments regarding the NIST approach to signature design based on our own experience of developing cryptographic algorithms and computer-based security systems.

We would like to begin by conveying a message of support and appreciation for the work NIST has undertaken to develop a much needed public key DSS, and say that the proposed algorithm appears most suitable for its alleged purpose.

In recognition and appreciation of the mission of NIST and its advisory board, the National Security Agency, we are sensitive to the motivations and interests that govern the design and implementation of any government-approved publicly available cryptographic scheme. Therefore, we have included as part of our response, a mathematical treatise which I believe aptly demonstrates our understanding of the proposed DSS strengths and, indeed, limitations. Also, any comment of the value or security offered by the DSS should, we feel, be guided by the overall system design, which would include message encryption, authentication, digital signature and a hashing function suitable to the overall intended security-level objectives. For this reason one might conclude the proposed DSS algorithm, when suitably coupled to an encryption scheme and compatible hashing function, could provide an entirely suitable methodology.

In view of the controversy over the widely used RSA system in regard to ownership, speed and compatibility with NIST objectives, we would like to bring to your attention a high-speed public key system already implemented in $GF(2)^{593}$ in combination with full DES functionality. Indeed, our signature scheme is very close to the proposed NIST standard, but different in the underlying algebraic method used. Compatible security is provided with this implementation. We would also bring to your attention that our use of a strong hashing function using discrete exponentiation as one component of the one-way function, also provides many user features and benefits.

Given that there may be some intellectual property issues surrounding the use of the concept of subgroup of order q, this issue could be avoided by broadening the scope of the NIST proposal to other algebraic systems, which might suggest that NIST review our current system and elliptic curve implementation which will be available shortly. In the spirit of this preceding point, our modification of ElGamal is a significant alteration of the ElGamal system and perhaps could diffuse patent claims based on Diffie-Hellman. Our implementation of ElGamal changes the security of the system from discrete logs to finding the solution of an exponential equation of the form $x^{f(x)} = \beta$. Our modification, in terms of security, relies on a more difficult problem than the discrete log problem; and more important, since it is not patented, is freely available. It is also more efficient than the current NIST proposal, saving one computation for finding an inverse and requiring only two modular multiplications.

As stated previously, we offer these comments in the hope there might be some flexibility in NIST's philosophy of how it might best implement a public key DSS. Moreover, we would request that some grace period be granted to companies who have made significant investment to bring public key systems to market, on the assumption of course, that NIST is comfortable with the algorithmic implementation. It would be our hope that NIST create a grandfather clause in its DSS proposal with alternate criteria for digital signature to qualified companies for some period of time, say two years, after which the DSS standard as specified would be fully implemented. Indeed, it would be comforting to see NIST choose a suitable algorithmic arrangement completely free of any intellectual property issues, for the benefit of all users now and in the future.

The Algorithm

Möbius Encryption Technologies has studied the (NIST DSS) document carefully and our overall impression is that the standard proposed is well thought out and is very practical. In order to discuss the digital signature algorithm, we will briefly describe it.

Let p and q be prime numbers so that $q|p - 1$ and p is approximately 512 bits and q is approximately 160 bits. Let β be a generating element for the cyclic subgroup of order q in $GF^*(p)$. For a an integer, $0 \leq a \leq q - 1$, β^a will be the public key and a the private key. Messages will be integers m where $0 \leq m \leq p - 1$.

It is assumed that a good hash function h(m) is available.

To create a signature:

- We first compute the hash h(m) of m in order to sign message m.
- Generate a random integer k and compute $r = \beta^k$.
- Solve the congruence for s:

$$h(m) \equiv -ar + ks \pmod q$$

The signature is the pair $(s, r \pmod q)$.

In order to check signatures:

- Compute $u = \beta^{h(m)}$.
- Compute $v = (\beta^a)^{r (\mathrm{mod}\ q)}$.
- Compute $w = uv^{s^{-1} (\mathrm{mod}\ q)}$.

The signature verifies if $w (\mathrm{mod}\ q) = r (\mathrm{mod}\ q)$.

Comments

- The signature is at most 320 bits in length. With the standard ElGamal scheme the full value of r (and not $r(\text{mod } q)$) would be required. This part of the new proposal is very clever and appealing. The signature could be shortened further if we replace $r(\text{mod } q)$ with the hash of r (provided the hash of r is less than 160 bits).

- The security of the scheme relies on the difficulty of computing discrete logarithms in a cyclic group of order q. Since this group is represented as the cyclic subgroup of order q contained in the multiplicative subgroup of the finite field of order p where p is about 512 bits long, then the security level is that required to compute logarithms in a field whose order is about 512 bits long.

- Working modulo *a* 512-bit modulus requires all exponentiations to be done with 512-bit numbers. This will affect the performance of the system and make hardware more complex.

- The signing procedure requires the computation of $k^{-1}(\text{mod } q)$ and the signature verification requires the computation of $s^{-1}(\text{mod } q)$. These are not particularly costly since they are modulo *a* 160-bit modulus, but the time requirement is nevertheless nonnegligible.

- The proposed scheme could be adapted to any cyclic group of order q and not just those embedded in the integers modulo p (provided of course that the group yields a suitable level of security). This will be considered later in greater detail.

Suggestions and Rationale

We also wanted to point out and emphasize the necessity of a good hash function for the proposed signature scheme. This is vital to the security and acceptance of the system. (The importance of a hash function is clearly stated in the NIST document.)

- Signature creation can be made more efficient by solving the modular equation

$$h(m) \equiv -as + kh(r)(\text{mod } q).$$

Since a is the fixed private key of the signer, a^{-1} can be computed once and for all signatures. This eliminates the need to compute k^{-1} each time a message is to be signed. The signature is still the pair $(s, h(r))$. Signature verification requires $h^{-1}(r)(\text{mod } q)$. The security of the system now relies partially on the difficulty of solving equations of the form $x^{h(x)}(\text{mod } q)$. This problem appears to be as difficult as the discrete logarithm problem itself.

- As mentioned, the hash function h can also be applied to r to reduce the signature's size.

Broadening the Scope

There are several reasons why NIST might consider broadening the scope of the proposed signature standard.

- the underlying algebraic system ($GF(p)$) is relatively slow.
- there are other algebraic systems which might be more efficient for various reasons.
- $GF(p)$ is more difficult to implement in hardware than other systems.
- since the intention of the proposed scheme is to do signing in a group whose order is 160 bits it seems reasonable and more efficient to also find an algebraic system (in this case a cyclic group whose order is about 160 bits) and all computations can be performed within the system. For example, the proposed scheme needs to do computations in a 512-bit system to obtain adequate security even though signing is in a much smaller group.

For clarity we describe a modification of the NIST scheme as applied to only a cyclic group of order q.

An Algorithm

Pick a cyclic group with q elements (q can be 150 to 160 bits) in which computation can be carried out efficiently and yet the discrete logarithm problem is difficult (eg., an elliptic curve system over a finite field would do). Let a be a random integer, α a generator for the group C_q and compute the public key α^a. Let h be a hash function from the message space to the integers from 1 to $q - 1$.

To create a signature:
- We first compute the hash $h(m)$ of m in order to sign message m.
- Generate a random integer k and compute $r = \alpha^k$.
- Solve the congruence for s:

$$h(m) \equiv -ar + ks \quad (\text{mod } q)$$

The signature is the pair (s, r).

To check signatures:
- Compute $u = \alpha^{h(m)}$.
- Compute $v = (\alpha^a)^r$.
- Compute $w = uv^{s^{-1}}$.

The signature verifies if $w = r$ (or $w = h(r)$ if we hash r).

Comments

- as mentioned earlier, one inverse computation can be eliminated by solving
$h(m) \equiv -as + kr(\text{mod } q)$ for s.
- the signer can compute $h^{-1}(r)(\text{mod } q)$ and send the signature $(s, h^{-1}(r))$. This saves the verifier from having to compute an inverse.

The Möbius Digital Signature System

The Möbius digital signature system, although different, is closely related to the NIST proposal.

The public key scheme used in the Trade Secrets product is based on the finite field $GF(2^{593})$. (The algebraic system can be extended to $GF(2^{1186})$ for additional security if desired.) The representation of the field is by a normal basis and the arithmetic processor is contained on a single VLSI device (Calmos part number CA34C168). The chip was designed by researchers at the University of Waterloo, and layed out by Calmos Semiconductor Inc. of Kanata, Ontario. When $GF(2^{593})$ is used the underlying algebraic system is the cyclic group of order $2^{593} - 1$ and if a quadratic extension (ie., $GF(2^{1186})$ were to be used, the system would be the cyclic group of order $2^{593} + 1$.

Messages in our system are binary 593-tuples. Let $f(x)$ be the hash value of the 593-tuple x obtained from the hash function f to be described next. Note that if a 593-tuple appears in a modular expression we will replace it by the integer it would represent if it were treated as an integer in its binary representation. For a user A with private key a and public key α^a to sign the 593-tuple m, A does the following:

- A generates a random integer k and computes $r = \alpha^k$. (Note that k is a binary vector of length 593 and that the exponentiation is done by the chip.)
- A solves the congruence

$$f(m) \equiv as + kf(r) (\text{mod } q)$$

for s where $q = 2^{593} - 1$. Notice that

$$s \equiv [f(m) - kf(r)]a^{-1} (\text{mod } q)$$

and so s is easily computed once a^{-1} is known.

- The signature is the pair (s, r). We note that s is a binary 593-tuple whereas r is at most a 593-tuple.

Let K be a 256-bit binary string, and let S be a 593-bit binary string (which is to be viewed as a field element in its coordinate form with respect to the normal basis N employed in the operation of the CA34C168 microprocessor used in Möbius products, henceforth referred to as the microprocessor). If K is placed in the R register and S is placed in the A and B registers of the microprocessor, then under the operation EXP2, K is expanded to a 593-bit exponent $K' = g(K)$ and the value $S^{g(K)}$ is computed by the device. The important facts about this from the point of view of the hash function is that $g(K)$ is nonlinear and the device does not admit $g(K) = 0$, which ensures that $S^{g(K)}$ is sensitive to S (see box for a detailed description of g as given in the design specification of the microprocessor).

Another primitive employed in the algorithm is best described in the accompanying diagram. In this primitive, a 256-bit binary string is placed in the register labeled *exponent register* which is best thought of as a shift register with feedback through the 1-bit adder. A second 593-bit string is placed in the register labeled *product register,* and the 256-bit string (denoted by $T + {}_2S$), is calculated by adding bitwise, using real integer arithmetic, the contents of the exponent register and the product register, with the resultant bit being fed back to the high-order bit of the exponent register, which rotates twice and then shifts another 81 positions during the operation (the final carry is ignored). Alternatively, given the 256-bit string $T = (T_{255}, T_{254}, \ldots, T_0)$ and the 593-bit string $S = (S_{592}, S_{591}, \ldots, S_0)$, then $T + {}_2S$ is defined by the following algorithm.

```
carry = 0
for q from 0 to 592
        $X = S_q + TQ + carry$
            if $X = 3$ then
                    carry = 1
                    else carry = 0
            endif
        $T_{256+q} = X$   (mod 2)
endf or
set $T + {}_2S = (T_{848}, T_{847}, \ldots, T_{593})$
end
```

The effect of this is if S is the concatenated string (S_2, S_1, S_0) where S_0 and S_1 are of length 81 bits, then $T + {}_2S$ is the integer sum

$$T = S_2 + S_1 + S_0 + e + f - g$$

where e, f and g are the carry bits after 256, 512 and 593 steps of the algorithm.

The hashing algorithm proceeds as follows. Let M be the message t to be hashed, presented as a binary string. Then M is parsed as the concatenated string

$$M = M'_1 M'_2 \ldots M'_m$$

where each of the M'_i, $i = 1, 2, \ldots, m$ is the length 592. We note that the last block of the message is "padded out" with "01010101 . . ." to give a length divisible by 592. Each M'_i is expanded to a block M_i of length 593 by adjoining to M'_i a new "least significant" bit which is the 0, 1 complement of the existing "least significant" bit of M'_i.

Let IV denote a randomly chosen but known 256-bit string or *Initialization Vector* (such a string is built into the microprocessor). For the first block of the message, form $K_1 = IV + {}_2M_1$, and $C_1 = M_1^{g(K_1)}$.

Subsequent message blocks are processed as:

$$K_i = K_{i-1} + {}_2C_{i-1} + {}_2M_i$$

$$C_i = M_i^{g(K_i)} = M_i^{g(K_{i-1} + {}_2C_{i-1} + {}_2M_i)}.$$

The hash value is C_m. This technique takes advantage of the difficulty of deriving solutions for equations of the type $x^{f(x)} = b$ over finite fields.

John C. Anderson
Möbius Encryption Technologies
Mississauga, Ontario L5R3L7
Canada

The instruction set for the CA34C168 microprocessor is:

```
                              Preload IV into R register
                              load into input register
Initialize R register         move $M_i$ to R
LOADD                         add $M_i$ to R
CP22AB                        form $C_i = M_i^{g(K_i)}$
CPA2R+                        check for end of message
EXP2                          calculate next $K_i$
TEST CBF, END IF = 1
CPA2R+
REPEAT TO LOAD
```

John C. Anderson is President and C.E.O. of Möbius Encryption Technologies, Ontario, Canada.

On May 7, 1992, NIST director John W. Lyons and James Burrows, director of NIST's Computer Systems Laboratory appeared before the Judiciary Committee of the House of Representatives to discuss information security. After summarizing the various types and uses of cryptography, Lyons turned to the current status of the DSS and addressed some of the comments and criticisms the agency has received. The following remarks have been gleaned from his prepared statement.

The proposed Digital Signature Standard, or DSS, was designed only to compute and verify digital signatures. It is different from, but complementary to, the DES, which is optimized for fast encryption of information. Furthermore, the DES is a cryptographic technique that uses one key to both encrypt and decrypt data. Therefore, two people seeking to communicate securely must have the same key. This was acceptable until a need for electronic funds transfer, electronic commerce and electronic key exchange indicated the desirability of a technique for which only *one* person had the key in order to create a message, so that anyone could verify its source and integrity but that *no one else* (including the intended recipient) could modify or create a message and claim it came from that person. Since a person's signature retains its characteristic and has been used for centuries to denote legal notification of document integrity and approval, the term "digital signature" was selected to denote legal notification of the integrity and approval of a message created on a digital computer.

A digital signature system is based on the fairly new field of public key cryptography. This system uses a computer program that implements a logical or mathematical method of combining a representation, or hash code, of a digital message with the private signature generation key of the originator of the message. A paired, but distinct, public signature verification key can be distributed to everyone because the private key cannot (if the technique is "secure") be computed from the public key. Therefore, one key is used to sign a message and a different key is used to verify it. While DES is called a one-key system, the DSS uses a two-key system.

In selecting the digital signature algorithm (DSA) for the proposed DSS, the following factors, among others, were considered: the level of security provided, the applicability of patents, the ease of export from the U.S. and the efficiency in a number of government and commercial applications and the impact on national security and law enforcement. A number of techniques were reviewed and deemed adequate to provide appropriate protection for federal systems. Among these, we placed primary emphasis on selecting technology that best assures appropriate security for federal information and did not require payment of royalties by U.S. private or commercial interests.

I would like to summarize the positive and negative comments we have received in response to our solicitation of comments.

Examples of a few of the positive comments include:
- The DSA will be especially useful to the financial services industry.
- There will be minimal cost impact if the proposed standard is implemented.
- Generating keys for the DSA is very fast.
- The DSA is the only signature algorithm that has been publicly proposed by any government.
- The algorithm should be adopted as a FIPS.

Many federal agencies and private organizations stated that a DSS was needed. Many believed the government's goal of having a standard that was free of patent impediments was good. However, the proposed DSS came under criticism, just as the DES had exactly 15 years earlier. The negative comments reflect some of the diverse interests in this area. Our responses follow each of the summarized comments.

- *The DSA selection process was not public.*

In response, NIST acted within its normal standards development procedure and the provisions of the Computer Security Act of 1987, drawing on the expertise of the National Security Agency, in proposing the DSS. In the normal standards development process, NIST identifies the need for a standard, produces technical specifications of a standard using inputs from different sources, and then solicits government and public comment on the proposal. After the comment period, the comments are analyzed, appropriate changes are made and a revised standard issued (or further comment is solicited if the revisions are substantial). This public process is being followed.

- *The DSA cannot be used for key distribution and encryption.*

In response, the DSA was designed to be a digital signature algorithm. It is not for key distribution or encryption. The DES is a much more efficient method of data encryption than available public key techniques in current implementations. A technique is needed for key distribution that is optimized to meet all federal cryptographic criteria.

- *The DSA does not provide adequate security.*

In response, what we are proposing provides excellent security. It provides orders of magnitude greater protection than is needed in the foreseeable future. NIST proposed a 512-bit modulus for efficiency reasons. Based on comments received and subsequent government analysis, NIST is proposing a number of modulus lengths between 512 bits and 1,024 bits. To counter the "trapdoor" allegations made against the DSS, NIST intends to include in a revised proposal a prime generation process which, together with the variable-length modulus, will mitigate many arguments about the security provided by the DSS. The DSS has undergone the same analysis procedures used for classified algorithms that are used for classified purposes and deemed appropriate for its intended uses.

- *The DSA is less efficient than the leading alternative.*

In response, the DSA was designed to satisfy many diverse and often conflicting criteria. It was difficult to meet, and impossible to optimize all of the criteria simultaneously. NIST was aware that the DSA is more efficient in computing signatures than in verifying them. We also were aware that it is very fast in generating the needed keys. Generating keys and signing must be efficient in some applications where limited computing capability may exist. For example, many users in the future will carry a smart card, similar in size to a credit card, but embodying a tiny computer that will generate personal signature keys and be able to "sign" electronic transactions for the owner. Prototype smart cards are in design now which sign in a second and verify in three seconds. We are aware of current DSS hardware implementations that sign and verify in milliseconds. The ability of the DSS to achieve greater efficiency through precomputation of some variables and through optimization techniques that are being developed will make the DSS highly desirable in many applications.

- *The DSA may infringe other patents.*

In response, a major criterion for the invention and selection of the DSS by the government was to avoid patented technology that could result in payment of royalties for government, commercial and private use. A patent has been applied for the DSA by the government with the intent of issuing licenses for this patent on a nonexclusive, royalty-free basis. Two patent holders claimed that the DSS infringes their patents. The infringement claims are still being reviewed.

- *The DSA is not compatible with formal and* de facto *standards.*

In response, the DSA is a new invention designed to meet a number of federal government criteria. It is proposed as a federal government standard for all unclassified applications, including Warner Amendment information. It is being proposed as a candidate American National Standard by a voluntary industry standards group. While not compatible with the leading *de facto* standard public key algorithm, the basic mathematical operations (exponentiation of large integers in a finite field) are identical and a large amount (60–80%) of the code needed to implement the two techniques is identical. The International Organization for Standardization is developing two generic digital signature standards, one which utilizes a hashing algorithm and one which does not. The DSA utilizes a hashing technique for efficiency purposes and can be proposed as a candidate algorithm for that generic standard.

Future Plans

NIST is working to resolve the negative comments and investigating a national infrastructure that will support digital signature applications in a cost-effective manner. Such an infrastructure is popularly called a Signature Certification Authority. We perceive a great need for such an infrastructure. Electronic filing of corporate and personal tax returns could be made more efficient and more secure if such a structure is available. Federal payments to contractors, vendors and social security recipients could be optimized if the integrity and authenticity of electronic payments could be assured. NIST is talking with federal organizations responsible for such large-scale applications, and we intend to hold a workshop with potential users to understand their requirements.

Specifically, we plan the following activities in adopting a DSS as a Federal Information Processing Standard (FIPS):

- Complete analysis and summary of public comments;
- Analyze and attempt to resolve patent issues;
- Initiate study of a signature certification authority infrastructure;
- Propose technical enhancements to DSA;
- Issue second solicitation comments on revised DSA (if needed);
- Hold a workshop on uses of the DSA;
- Investigate the economic stakes and interest involving the DSS;
- Coordinate and harmonize the revised DSS with ANSI and ISO standards activites;
- Conduct final coordination of DSS within government;
- Recommend DOC approval of DSS;
- Publish revised DSS after DOC approval.

Cryptography has the potential of being a major enabling technology in electronic commerce. Cryptography is required to protect electronic information in modern, distributed computer and communications systems. There is no other way of assuring integrity and confidentiality of information in national or international information communications. Cryptographic technology is needed which assures cost-effective government and commercial security and which preserves legitimate national security and law enforcement interests.

ACM 0002-0782/92/0700-032