CHAPTER 20

Public-Key Digital Signature Algorithms

20.1 DIGITAL SIGNATURE ALGORITHM (DSA)

In August 1991, The National Institute of Standards and Technology (NIST) proposed the Digital Signature Algorithm (DSA) for use in their Digital Signature Standard (DSS). According to the *Federal Register* [538]:

> A Federal Information Processing Standard (FIPS) for Digital Signature Standard (DSS) is being proposed. This proposed standard specifies a public-key digital signature algorithm (DSA) appropriate for Federal digital signature applications. The proposed DSS uses a public key to verify to a recipient the integrity of data and identity of the sender of the data. The DSS can also be used by a third party to ascertain the authenticity of a signature and the data associated with it.
>
> This proposed standard adopts a public-key signature scheme that uses a pair of transformations to generate and verify a digital value called a signature.

And:

> This proposed FIPS is the result of evaluating a number of alternative digital signature techniques. In making the selection NIST has followed the mandate contained in section 2 of the Computer Security Act of 1987 that NIST develop standards to ". . . assure the cost-effective security and privacy of Federal information and, among technologies offering comparable protection, on selecting the option with the most desirable operating and use characteristics."
>
> Among the factors that were considered during this process were the level of security provided, the ease of implementation in both hardware and software, the ease of export from the U.S., the applicability of patents, impact on national security and law enforcement and the level of efficiency in both the signing and verification functions. A number of techniques were deemed to provide appropriate protection for Federal systems. The technique selected has the following desirable characteristics:

> NIST expects it to be available on a royalty-free basis. Broader use of this technique resulting from public availability should be an economic benefit to the government and the public.
>
> The technique selected provides for efficient implementation of the signature operations in smart card applications. In these applications the signing operations are performed in the computationally modest environment of the smart card while the verification process is implemented in a more computationally rich environment such as a personal computer, a hardware cryptographic module, or a mainframe computer.

Before it gets too confusing, let me review the nomenclature: DSA is the algorithm; the DSS is the standard. The standard employs the algorithm. The algorithm is part of the standard.

Reaction to the Announcement

NIST's announcement created a maelstrom of criticisms and accusations. Unfortunately, it was more political than academic. RSA Data Security, Inc., purveyors of the RSA algorithm, led the criticism against DSS. They wanted RSA, and not another algorithm, used as the standard. RSADSI makes a lot of money licensing the RSA algorithm, and a royalty-free digital signature standard would directly affect their bottom line. (Note: DSA is not necessarily free of patent infringements; I'll discuss that later.)

Before the algorithm was announced, RSADSI campaigned against a "common modulus," which might have given the government the ability to forge signatures. When the algorithm was announced without this common modulus, they attacked it on other grounds [154], both in letters to NIST and statements to the press. (Four letters to NIST appeared in [1326]. When reading them, keep in mind that at least two of the authors, Rivest and Hellman, had a financial interest in DSS's not being approved.)

Many large software companies that already licensed the RSA algorithm came out against the DSS. In 1982, the government had solicited public-key algorithms for a standard [537]. After that, there wasn't a peep out of NIST for nine years. Companies such as IBM, Apple, Novell, Lotus, Northern Telecom, Microsoft, DEC, and Sun had already spent large amounts of money implementing the RSA algorithm. They were not interested in losing their investment.

In all, NIST received 109 comments by the end of the first comment period on February 28, 1992.

Let's look at the criticisms against DSA, one by one.

1. DSA cannot be used for encryption or key distribution.

   True, but not the point of the standard. This is a signature standard. NIST should have a standard for public-key encryption. NIST is committing a grave injustice to the American people by not implementing a public-key

encryption standard. It is suspicious that this proposed digital signature standard cannot be used for encryption. (As it turns out, though, it can—see Section 23.3.) That does not mean that a signature standard is useless.

2. DSA was developed by the NSA, and there may be a trapdoor in the algorithm.

   Much of the initial comments were just paranoia: "NIST's denial of information with no apparent justification does not inspire confidence in DSS, but intensifies concern that there is a hidden agenda, such as laying the groundwork for a national public-key cryptosystem that is in fact vulnerable to being broken by NIST and/or NSA" [154]. One serious question about the security of DSA was raised by Arjen Lenstra and Stuart Haber at Bellcore. This will be discussed later.

3. DSA is slower than RSA [800].

   True, more or less. Signature generation speeds are the same, but signature verification can be 10 to 40 times slower with DSA. Key generation, however, is faster. But key generation is irrelevant; a user rarely does it. On the other hand, signature verification is the most common operation.

   The problem with this criticism is that there are many ways to play with the test parameters, depending on the results you want. Precomputations can speed up DSA signature generation, but don't always apply. Proponents of RSA use numbers optimized to make their calculations easier; proponents of DSA use their own optimizations. In any case, computers are getting faster all the time. While there is a speed difference, it will not be noticeable in most applications.

4. RSA is a *de facto* standard.

   Here are two examples of this complaint. From Robert Follett, the program director of standards at IBM [570]:

   > IBM is concerned that NIST has proposed a standard with a different digital signature scheme rather than adopting the international standard. We have been convinced by users and user organizations that the international standards using RSA will be a prerequisite to the sales of security products in the very near future.

   From Les Shroyer, vice president and director, corporate MIS and telecommunications, at Motorola [1444]:

   > We must have a single, robust, politically-accepted digital signature standard that is usable throughout the world, between both U.S. and non-U.S., and Motorola and non-Motorola entities. The lack of other viable digital signature technology for the last eight years has made RSA a de facto standard. . . . Motorola and many other companies . . . have committed millions of dollars to RSA. We have concern over the interoperability and support of two different standards, as that situation will lead to added costs, delays in deployment, and complication. . . .

Many companies wanted NIST to adopt the ISO 9796, the international digital signature standard that uses RSA [762]. While this is a valid complaint, it is not a sufficient justification to make it a standard. A royalty-free standard would better serve the U.S. public interest.

5. The DSA selection process was not public; sufficient time for analysis has not been provided.

   First NIST claimed that they designed the DSA; then they admitted that NSA helped them. Finally, they confirmed that NSA designed the algorithm. This worries many people; the NSA doesn't inspire trust. Even so, the algorithm is public and available for analysis; and NIST extended the time for analysis and comment.

6. DSA may infringe on other patents.

   It may. This will be discussed in the section on patent issues.

7. The key size is too small.

   This was the only valid criticism of DSS. The original implementation set the modulus at 512 bits [1149]. Since the algorithm gets its security from the difficulty of computing discrete logs in that modulus, this worried most cryptographers. There have since been advances in the problem of calculating discrete logarithms in a finite field, and 512 bits is too short for long-term security (see Section 7.2). According to Brian LaMacchia and Andrew Odlyzko, "... even 512-bit primes appear to offer only marginal security ..." [934]. In response to this criticism, NIST made the key size variable, from 512 bits to 1024 bits. Not great, but better.

On May 19, 1994, the standard was finally issued [1154]. The issuing statement said [542]:

> This standard is applicable to all Federal departments and agencies for the protection of unclassified information. ... This standard shall be used in designing and implementing public-key based signature schemes which Federal departments and agencies operate or which are operated for them under contract. Adoption and use of this standard is available to private and commercial organizations.

Before you run out and implement this standard in your next product, read the section on patent issues below.

Description of DSA

DSA is a variant of the Schnorr and ElGamal signature algorithms, and is fully described in [1154]. The algorithm uses the following parameters:

> p = a prime number L bits long, when L ranges from 512 to 1024 and is a multiple of 64. (In the original standard, the size of p was fixed at 512 bits [1149]. This was the source of much criticism and was changed by NIST [1154].)

q = a 160-bit prime factor of $p - 1$.

$g = h^{(p-1)/q} \bmod p$, where h is any number less than $p - 1$ such that $h^{(p-1)/q} \bmod p$ is greater than 1.

x = a number less than q.

$y = g^x \bmod p$.

The algorithm also makes use of a one-way hash function: $H(m)$. The standard specifies the Secure Hash Algorithm, discussed in Section 18.7.

The first three parameters, p, q, and g, are public and can be common across a network of users. The private key is x; the public key is y.

To sign a message, m:

(1) Alice generates a random number, k, less than q.

(2) Alice generates

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

The parameters r and s are her signature; she sends these to Bob.

(3) Bob verifies the signature by computing

$$w = s^{-1} \bmod q$$
$$u_1 = (H(m) * w) \bmod q$$
$$u_2 = (rw) \bmod q$$
$$v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$$

If $v = r$, then the signature is verified.

Proofs for the mathematical relationships are found in [1154]. Table 20.1 provides a summary.

Speed Precomputations

Table 20.2 gives sample software speeds of DSA [918].

Real-world implementations of DSA can often be speeded up through precomputations. Notice that the value r does not depend on the message. You can create a string of random k values, and then precompute r values for each of them. You can also precompute k^{-1} for each of those k values. Then, when a message comes along, you can compute s for a given r and k^{-1}.

This precomputation speeds up DSA considerably. Table 20.3 is a comparison of DSA and RSA computation times for a particular smart card implementation [1479].

Table 20.1
DSA Signatures

Public Key:

p 512-bit to 1024-bit prime (can be shared among a group of users)
q 160-bit prime factor of $p - 1$ (can be shared among a group of users)
g $= h^{(p-1)/q} \bmod p$, where h is less than $p - 1$ and $h^{(p-1)/q} \bmod p > 1$ (can be shared among a group of users)
y $= g^x \bmod p$ (a p-bit number)

Private Key:

x $< q$ (a 160-bit number)

Signing:

k choose at random, less than q
r (signature) $= (g^k \bmod p) \bmod q$
s (signature) $= (k^{-1} (H(m) + xr)) \bmod q$

Verifying:

$w = s^{-1} \bmod q$
$u_1 = (H(m) * w) \bmod q$
$u_2 = (rw) \bmod q$
$v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$
If $v = r$, then the signature is verified.

DSA Prime Generation

Lenstra and Haber pointed out that certain moduli are much easier to crack than others [950]. If someone forced a network to use one of these "cooked" moduli, then their signatures would be easier to forge. This isn't a problem for two reasons: These moduli are easy to detect and they are so rare that the chances of using one when choosing a modulus randomly are almost negligible—smaller, in fact, than the chances of accidentally generating a composite number using a probabilistic prime generation routine.

In [1154] NIST recommended a specific method for generating the two primes, p and q, where q divides $p - 1$. The prime p is L bits long, between 512 and 1024 bits

Table 20.2
DSA Speeds for Different Modulus Lengths with a 160-bit Exponent (on a SPARC II)

	512 bits	768 bits	1024 bits
Sign	0.20 sec	0.43 sec	0.57 sec
Verify	0.35 sec	0.80 sec	1.27 sec

Table 20.3
Comparison of RSA and DSA Computation Times

	DSA	RSA	DSA with Common p, q, g
Global Computations	Off-card (P)	N/A	Off-card (P)
Key Generation	14 sec	Off-card (S)	4 sec
Precomputation	14 sec	N/A	4 sec
Signature	.03 sec	15 sec	.03 sec
Verification	16 sec	1.5 sec	10 sec
	1–5 sec off-card (P)	1–3 sec off-card (P)	

Off-card computations were performed on an 80386 33 mHz, personal computer. (P) indicates public parameters off-card and (S) indicates secret parameters off-card. Both algorithms use a 512-bit modulus.

long, in some multiple of 64 bits. The prime q is 160 bits long. Let $L - 1 = 160n + b$, where L is the length of p, and n and b are two numbers and b is less than 160.

(1) Choose an arbitrary sequence of at least 160 bits and call it S. Let g be the length of S in bits.

(2) Compute $U = \text{SHA}(S) \oplus \text{SHA}((S + 1) \bmod 2^g)$, where SHA is the Secure Hash Algorithm (see Section 18.7).

(3) Form q by setting the most significant bit and the least significant bit of U to 1.

(4) Check whether q is prime.

(5) If q is not prime, go back to step (1).

(6) Let $C = 0$ and $N = 2$.

(7) For $k = 0, 1, \ldots, n$, let $V_k = \text{SHA}((S + N + k) \bmod 2^g)$

(8) Let W be the integer

$$W = V_0 + 2^{160}V_1 + \ldots + 2^{160(n-1)}V_{n-1} + 2^{160n}(V_n \bmod 2^b)$$

and let

$$X = W + 2^{L-1}$$

Note that X is an L-bit number.

(9) Let $p = X - ((X \bmod 2q) - 1)$. Note that p is congruent to 1 mod $2q$.

(10) If $p < 2^{L-1}$, then go to step (13).

(11) Check whether p is prime.

(12) If p is prime, go to step (15).

(13) Let $C = C + 1$ and $N = N + n + 1$.

(14) If $C = 4096$, then go to step (1). Otherwise, go to step (7).

(15) Save the value of S and the value of C used to generate p and q.

In [1154], the variable S is called the "seed," C is called the "counter," and N the "offset."

The point of this exercise is that there is a public means of generating p and q. For all practical purposes, this method prevents cooked values of p and q. If someone hands you a p and a q, you might wonder where that person got them. However, if someone hands you a value for S and C that generated the random p and q, you can go through this routine yourself. Using a one-way hash function, SHA in the standard, prevents someone from working backwards from a p and q to generate an S and C.

This security is better than what you get with RSA. In RSA, the prime numbers are kept secret. Someone could generate a fake prime or one of a special form that makes factoring easier. Unless you know the private key, you won't know that. Here, even if you don't know a person's private key, you can confirm that p and q have been generated randomly.

ElGamal Encryption with DSA

There have been allegations that the government likes the DSA because it is only a digital signature algorithm and can't be used for encryption. It is, however, possible to use the DSA function call to do ElGamal encryption.

Assume that the DSA algorithm is implemented with a single function call:

```
DSAsign (p,q,g,k,x,h,r,s)
```

You supply the numbers p, q, g, k, x, and h, and the function returns the signature parameters: r and s.

To do ElGamal encryption of message m with public key y, choose a random number, k, and call

```
DSAsign (p,p,g,k,0,0,r,s)
```

The value of r returned is a in the ElGamal scheme. Throw s away. Then, call

```
DSAsign (p,p,y,k,0,0,r,s)
```

Rename the value of r to be u; throw s away. Call

```
DSAsign (p,p,m,1,u,0,r,s)
```

Throw r away. The value of s returned is b in the ElGamal scheme. You now have the ciphertext, a and b.

Decryption is just as easy. Using secret key x, and ciphertext messages a and b, call

```
DSAsign (p,p,a,x,0,0,r,s)
```

The value r is $a^x \bmod p$. Call that e. Then call

```
DSAsign (p,p,1,e,b,0,r,s)
```

The value s is the plaintext message, m.

This method will not work with all implementations of DSA. Some may fix the values of p and q, or the lengths of some of the other parameters. Still, if the implementation is general enough, this is a way to encrypt using nothing more than digital signature function.

RSA Encryption with DSA

RSA encryption is even easier. With a modulus n, message m, and public key e, call

```
DSAsign (n,n,m,e,0,0,r,s)
```

The value of r returned is the ciphertext.

RSA decryption is the same thing. If d is the private key, then

```
DSAsign (n,n,m,d,0,0,r,s)
```

returns the plaintext as the value of r.

Security of DSA

At 512-bits, DSA wasn't strong enough for long-term security. At 1024 bits, it is.

The NSA, in its first public interview on the subject, commented to Joe Abernathy of *The Houston Chronicle* on allegations about a trapdoor in DSS [363]:

> Regarding the alleged trapdoor in the DSS. We find the term trapdoor somewhat misleading since it implies that the messages sent by the DSS are encrypted and with access via a trapdoor one could somehow decrypt (read) the message without the sender's knowledge.
>
> The DSS does not encrypt any data. The real issue is whether the DSS is susceptible to someone forging a signature and therefore discrediting the entire system. We state categorically that the chances of anyone—including NSA—forging a signature with the DSS when it is properly used and implemented is infinitesimally small.
>
> Furthermore, the alleged trapdoor vulnerability is true for *any* public key-based authentication system, including RSA. To imply somehow that this only affects the DSS (a popular argument in the press) is totally misleading. The issue is one of implementation and how one goes about selecting prime numbers. We call your attention to a recent EUROCRYPT conference which had a panel discussion on the issue of trapdoors in the DSS. Included on the panel was one of the Bellcore researchers who initially raised the trapdoor allegation, and our understanding is that the panel—including the person from Bellcore—concluded that the alleged trapdoor was not an issue for the DSS. Furthermore, the general consensus appeared to be that the trapdoor issue was trivial and had been overblown in the

press. However, to try to respond to the trapdoor allegation, at NIST's request, we have designed a prime generation process which will ensure that one can avoid selection of the relatively few weak primes which could lead to weakness in using the DSS. Additionally, NIST intends to allow for larger modulus sizes up to 1024 which effectively negates the need to even use the prime generation process to avoid weak primes. An additional very important point that is often overlooked is that with the DSS the primes are *public* and therefore can be subject to public examination. Not all public key systems provide for this same type of examination.

The integrity of any information security system requires attention to proper implementation. With the myriad of vulnerabilities possible given the differences among users, NSA has traditionally insisted on centralized trusted centers as a way to minimize risk to the system. While we have designed technical modifications to the DSS to meet NIST's requests for a more decentralized approach, we still would emphasize that portion of the *Federal Register* notice for the DSS which states:

> "While it is the intent of this standard to specify general security requirements for generating digital signatures, conformance to this standard does not assure that a particular implementation is secure. The responsible authority in each agency or department shall assure that an overall implementation provides an acceptable level of security. NIST will be working with government users to ensure appropriate implementations."

Finally, we have read all the arguments purporting insecurities with the DSS, and we remain unconvinced of their validity. The DSS has been subjected to intense evaluation within NSA which led to its being endorsed by our Director of Information Systems Security for use in signing unclassified data processed in certain intelligence systems and even for signing classified data in selected systems. We believe that this approval speaks to the lack of any credible attack on the integrity provided by the DSS given proper use and implementation. Based on the technical and security requirements of the U.S. government for digital signatures, we believe the DSS is the best choice. In fact, the DSS is being used in a pilot project for the Defense Message System to assure the authenticity of electronic messages of vital command and control information. This initial demonstration includes participation from the Joint Chiefs of Staff, the military services, and Defense Agencies and is being done in cooperation with NIST.

I'm not going to comment on the trustworthiness of the NSA. Take their comments for what you think they're worth.

Attacks against k

Each signature requires a new value of k, and that value must be chosen randomly. If Eve ever recovers a k that Alice used to sign a message, perhaps by exploiting some properties of the random-number generator that generated k, she can recover Alice's private key, x. If Eve ever gets two messages signed using the same k, even if she doesn't know what it is, she can recover x. And with x, Eve can generate undetectable forgeries of Alice's signature. In any implementation of the DSA, a good random-number generator is essential to the system's security [1468].

Dangers of a Common Modulus

Even though the DSS does not specify a common modulus to be shared by everyone, different implementations may. For example, the Internal Revenue Service is considering using the DSS for the electronic submission of tax returns. What if they require every taxpayer in the country to use a common p and q? Even though the standard doesn't require a common modulus, such an implementation accomplishes the same thing. A common modulus too easily becomes a tempting target for cryptanalysis. It is still too early to tell much about different DSS implementations, but there is some cause for concern.

Subliminal Channel in DSA

Gus Simmons discovered a subliminal channel in DSA [1468,1469] (see Section 23.3). This subliminal channel allows someone to embed a secret message in his signature that can only be read by another person who knows the key. According to Simmons, it is a "remarkable coincidence" that the "apparently inherent shortcomings of subliminal channels using the ElGamal scheme can all be overcome" in the DSS, and that the DSS "provides the most hospitable setting for subliminal communications discovered to date." NIST and NSA have not commented on this subliminal channel; no one knows if they even knew about it. Since this subliminal channel allows an unscrupulous implementer of DSS to leak a piece of the private key with each signature, it is important to never use an implementation of DSS if you don't trust the implementer.

Patents

David Kravitz, formerly of the NSA, holds a patent on DSA [897]. According to NIST [538]:

> NIST intends to make this DSS technique available world-wide on a royalty-free basis to the public interest. We believe this technique is patentable and that no other patents would apply to the DSS, but we cannot give firm assurances to such effect in advance of issuance of the patent.

Even so, three patent holders claim that the DSA infringes on their patents: Diffie-Hellman (see Section 22.1) [718], Merkle-Hellman (see Section 19.2) [720], and Schnorr (see Section 21.3) [1398]. The Schnorr patent is the most troublesome. The other two patents expire in 1997; the Schnorr patent is valid until 2008. The Schnorr algorithm was not developed with government money; unlike the PKP patents, the U.S. government has no rights to the Schnorr patent; and Schnorr patented his algorithm worldwide. Even if the U.S. courts rule in favor of DSA, it is unclear what other courts around the world would do. Is an international company going to adopt a standard that may be legal in some countries but infringes on a patent in others? This issue will take time to resolve; at the time of this writing it isn't even resolved in the United States.

In June 1993 NIST proposed to give PKP an exclusive patent license to DSA [541]. The agreement fell through after public outcry and the standard was issued without any deal. NIST said [542]:

> . . . NIST has addressed the possible patent infringement claims, and has concluded that there are no valid claims.

So the standard is official, lawsuits are threatened, and no one knows what to do. NIST has said that it would help defend people sued for patent infringement, if they were using DSA to satisfy a government contract. Everyone else, it seems, is on their own. ANSI has a draft banking standard that uses DSA [60]. NIST is working to standardize DSA within the government. Shell Oil has made DSA their international standard. I know of no other proposed DSA standards.

20.2 DSA VARIANTS

This variant makes computation easier on the signer by not forcing him to compute k^{-1} [1135]. All the parameters are as in DSA. To sign a message, m, Alice generates two random numbers, k and d, both less than q. The signature is

$$r = (g^k \bmod p) \bmod q$$
$$s = (H(m) + xr) \star d \bmod q$$
$$t = kd \bmod q$$

Bob verifies the signature by computing

$$w = t/s \bmod q$$
$$u_1 = (H(m) \star w) \bmod q$$
$$u_2 = (rw) \bmod q$$

If $r = ((g^{u1} \star y^{u2}) \bmod p) \bmod q$, then the signature is verified.

This next variant makes computation easier on the verifier [1040,1629]. All the parameters are as in DSA. To sign a message, m, Alice generates a random number, k, less than q. The signature is

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

Bob verifies the signature by computing

$$u_1 = (H(m) \star s) \bmod q$$
$$u_2 = (sr) \bmod q$$

If $r = ((g^{u1} \star y^{u2}) \bmod p) \bmod q$, then the signature is verified.

Another DSA variant allows for batch verification; Bob can verify signatures in batches [1135]. If they are all valid, he is done. If one isn't valid, then he still has to find it. Unfortunately, it is not secure; either the signer or the verifier can easily create a set of bogus signatures that satisfy the batch criteria [974].

There is also a variant for DSA prime generation, one that embeds q and the parameters used to generate the primes within p. Whether this scheme reduces the security of DSA is still unknown.

(1) Choose an arbitrary sequence of at least 160 bits and call it S. Let g be the length of S in bits.

(2) Compute $U = \text{SHA}(S) \oplus \text{SHA}\ ((S + 1) \bmod 2^g)$, where SHA is the Secure Hash Algorithm (see Section 18.7).

(3) Form q by setting the most significant bit and the least significant bit of U to 1.

(4) Check whether q is prime.

(5) Let p be the concatenation of q, S, C, and SHA(S). C is set to 32 zero bits.

(6) $p = p - (p \bmod q) + 1$.

(7) $p = p + q$.

(8) If the C in p is 0x7fffffff, go to step (1).

(9) Check whether p is prime.

(10) If p is composite, go to step (7).

The neat thing about this variant is that you don't have to store the values of C and S used to generate p and q; they are embedded within p. For applications without a whole lot of memory, like smart cards, this can be a big deal.

20.3 GOST Digital Signature Algorithm

This is a Russian digital signature standard, officially called GOST R 34.10-94 [656]. The algorithm is very similar to DSA, and uses the following parameters

p = a prime number, either between 509 and 512 bits long, or between 1020 and 1024 bits long.

q = a 254- to 256-bit prime factor of $p - 1$.

a = any number less than $p - 1$ such that $a^q \bmod p = 1$.

x = a number less than q.

$y = a^x \bmod p$.

The algorithm also makes use of a one-way hash function: $H(x)$. The standard specifies GOST R 34.11-94 (see Section 18.11), a function based on the GOST symmetric algorithm (see Section 14.1) [657].

The first three parameters, p, q, and a, are public and can be common across a network of users. The private key is x; the public key is y.

To sign a message, m

(1) Alice generates a random number, k, less than q

(2) Alice generates

$$r = (a^k \bmod p) \bmod q$$
$$s = (xr + k(H(m))) \bmod q$$

If $H(m) \bmod q = 0$, then set it equal to 1. If $r = 0$, then choose another k and start again. The signature is two numbers: $r \bmod 2^{256}$ and $s \bmod 2^{256}$. She sends these to Bob.

(3) Bob verifies the signature by computing

$$v = H(m)^{q-2} \bmod q$$
$$z_1 = (sv) \bmod q$$
$$z_2 = ((q - r) * v) \bmod q$$
$$u = ((a^{z_1} * y^{z_2}) \bmod p) \bmod q$$

If $u = r$, then the signature is verified.

The difference between this scheme and DSA is that with DSA $s = (xr + k^{-1}(H(m))) \bmod q$, which leads to a different verification equation. Curious, though, is that q is 256 bits. Most Western cryptographers seem satisfied with a q of around 160 bits. Perhaps this is just a reflection of the Russian tendency to play it ultrasafe.

The standard has been in use since the beginning of 1995, and is not classified "for special use"—whatever that means.

20.4 DISCRETE LOGARITHM SIGNATURE SCHEMES

ElGamal, Schnorr (see Section 21.3), and DSA signature schemes are very similar. In fact, they are just three examples of a general digital signature scheme based on the Discrete Logarithm Problem. Along with thousands of other signature schemes, they are part of the same family [740,741,699,1184].

Choose p, a large prime number, and q, either $p - 1$ or a large prime factor of $p - 1$. Then choose g, a number between 1 and p such that $g^q \equiv 1 \pmod p$. All these numbers are public, and can be common to a group of users. The private key is x, less than q. The public key is $y = g^x \bmod p$.

To sign a message, m, first choose a random k less than and relatively prime to q. If q is also prime, any k less than q works. First compute

$$r = g^k \bmod p$$

The generalized **signature equation** now becomes

$$ak = b + cx \bmod q$$

The coefficients a, b, and c can be any of a variety of things. Each line in Table 20.4 gives six possibilities.

To verify the signature, the receiver must confirm that

$$r^a = g^b y^c \bmod p$$

This is called the **verification equation**.

Table 20.5 lists the signature and verifications possible from just the first line of potential values for a, b, and c, ignoring the effects of the ±.

Table 20.4
Possible Permutations of a, b, and c ($r' = r \bmod q$)

$\pm r'$	$\pm s$	m
$\pm r'm$	$\pm s$	1
$\pm r'm$	$\pm ms$	1
$\pm mr'$	$\pm r's$	1
$\pm ms$	$\pm r's$	1

That's six different signature schemes. Adding the negative signs brings the total to 24. Using the other possible values listed for a, b, and c brings the total to 120.

ElGamal [518,519] and DSA [1154] are essentially based on equation (4). Other schemes are based on equation (2) [24,1629]. Schnorr [1396,1397] is closely related to equation (5), as is another scheme [1183]. And equation (1) can be modified to yield the scheme proposed in [1630]. The rest of the equations are new.

There's more. You can make any of these schemes more DSA-like by defining r as

$$r = (g^k \bmod p) \bmod q$$

Keep the same signature equation and make the verification equation

$$u_1 = a^{-1}b \bmod q$$
$$u_2 = a^{-1}c \bmod q$$
$$r = (g^{u_1}y^{u_2} \bmod p) \bmod q$$

There are two other possibilities along these lines [740,741]; you can do this with each of the 120 schemes, bringing the total to 480 discrete-logarithm-based digital signature schemes.

But wait—there's more. Additional generalizations and variations can generate more than 13,000 variants (not all of them terribly efficient) [740,741].

One of the nice things about using RSA for digital signatures is a feature called **message recovery**. When you verify an RSA signature you compute m. Then you compare the computed m with the message and see if the signature is valid for that

Table 20.5
Discrete Logarithm Signature Schemes

Signature Equation	Verification Equation
(1) $r'k = s + mx \bmod q$	$r^{r'} = g^s y^m \bmod p$
(2) $r'k = m + sx \bmod q$	$r^{r'} = g^m y^s \bmod p$
(3) $sk = r' + mx \bmod q$	$r^s = g^{r'} y^m \bmod p$
(4) $sk = m + r'x \bmod q$	$r^s = g^m y^{r'} \bmod p$
(5) $mk = s + r'x \bmod q$	$r^m = g^s y^{r'} \bmod p$
(6) $mk = r' + sx \bmod q$	$r^m = g^{r'} y^s \bmod p$

message. With the previous schemes, you can't recover m when you compute the signature; you need a candidate m that you use in a verification equation. Well, as it turns out it is possible to construct a message recovery variant for all the above signature schemes.

To sign, first compute

$$r = mg^k \bmod p$$

and replace m by 1 in the signature equation. Then you can reconstruct the verification equation such that m can be computed directly.

You can do the same with the DSA-like schemes:

$$r = (mg^k \bmod p) \bmod q$$

All the variants are equally secure, so it makes sense to choose a scheme that is easy to compute with. The requirement to compute inverses slows most of these schemes. As it turns out, a scheme in this pile allows computing both the signature equation and the verification equation without inverses and also gives message recovery. It is called the **p-NEW** scheme [1184].

$$r = mg^{-k} \bmod p$$
$$s = k - r'x \bmod q$$

And m is recovered (and the signature verified) by

$$m = g^s y^{r'} r \bmod p$$

Some variants sign two and three message blocks at the same time [740]; other variants can be used for blind signatures [741].

This is a remarkable piece of research. All of the various discrete-logarithm-based digital signature schemes have been put in one coherent framework. In my opinion this finally puts to rest any patent dispute between Schnorr [1398] and DSA [897]: DSA is not a derivative of Schnorr, nor even of ElGamal. All three are examples of this general construction, and this general construction is unpatented.

20.5 ONG-SCHNORR-SHAMIR

This signature scheme uses polynomials modulo n [1219,1220]. Choose a large integer n (you need not know the factorization of n). Then choose a random integer, k, such that k and n are relatively prime. Calculate h such that

$$h = -k^{-2} \bmod n = -(k^{-1})^2 \bmod n$$

The public key is h and n; k is the private key.

To sign a message, M, first generate a random number, r, such that r and n are relatively prime. Then calculate:

$$S_1 = 1/2 * (M/r + r) \bmod n$$
$$S_2 = k/2 * (M/r - r) \bmod n$$

The pair, S_1 and S_2, is the signature.

To verify a signature, confirm that

$$S_1^2 + h * S_2^2 \equiv M \pmod{n}$$

The version of the scheme described here is based on quadratic polynomials. When it was first proposed in [1217], a $100 reward was offered for successful cryptanalysis. It was proved insecure [1255,18], but its authors were not deterred. They proposed a modification of the algorithm based on cubic polynomials, which is also insecure [1255]. The authors then proposed a quartic version, which was also broken [524,1255]. A variant which fixes these problems is in [1134].

20.6 ESIGN

ESIGN is a digital signature scheme from NTT Japan [1205,583]. It is touted as being at least as secure and considerably faster than either RSA or DSA, with similar key and signature lengths.

The private key is a pair of large prime numbers, p and q. The public key is n, when

$$n = p^2q$$

H is a hash function that operates on a message, m, such that $H(m)$ is between 0 and $n - 1$. There is also a security parameter, k, which will be discussed shortly.

(1) Alice picks a random number x, where x is less than pq.

(2) Alice computes:

w, the least integer that is larger than or equal to

$(H(m) - x^k \bmod n)/pq$

$s = x + ((w/kx^{k-1}) \bmod p)pq$

(3) Alice sends s to Bob.

(4) To verify the signature, Bob computes $s^k \bmod n$. He also computes a, which is the least integer larger than or equal to two times the number of bits of n divided by 3. If $H(m)$ is less than or equal to $s^k \bmod n$, and if $s^k \bmod n$ is less than $H(m) + 2^a$, then the signature is considered valid.

This algorithm works faster with precomputation. This precomputation can be done at any time and has nothing to do with the message being signed. After picking x, Alice could break step (2) into two partial steps. The first can be precomputed.

(2a) Alice computes:

$u = x^k \bmod n$

$v = 1/(kx^{k-1}) \bmod p$

(2b) Alice computes:

w = the least integer that is larger than or equal to
$(H(m) - u)/pq)$
$s = x + (wv \bmod p)pq$

For the size of numbers generally used, this precomputation speeds up the signature process by a factor of 10. Almost all the hard work is done in the precomputation stage. A discussion of modular arithmetic operations to speed ESIGN can be found in [1625,1624]. This algorithm can also be extended to work with elliptic curves [1206].

Security of ESIGN

When this algorithm was originally proposed, k was set to 2 [1215]. This was quickly broken by Ernie Brickell and John DeLaurentis [261], who then extended their attack to $k = 3$. A modified version of this algorithm [1203] was broken by Shamir [1204]. The variant proposed in [1204] was broken in [1553]. ESIGN is the current incarnation of this family of algorithms. Another new attack [963] does not work against ESIGN.

The authors currently recommend these values for k: 8, 16, 32, 64, 128, 256, 512, and 1024. They also recommend that p and q each be of at least 192 bits, making n at least 576 bits long. (I think n should be twice that length.) With these parameters, the authors conjecture that ESIGN is as secure as RSA or Rabin. And their analysis shows favorable speed comparison to RSA, ElGamal, and DSA [582].

Patents

ESIGN is patented in the United States [1208], Canada, England, France, Germany, and Italy. Anyone who wishes to license the algorithm should contact Intellectual Property Department, NTT, 1–6 Uchisaiwai-cho, 1-chome, Chiyada-ku, 100 Japan.

20.7 CELLULAR AUTOMATA

A new and novel idea, studied by Papua Guam [665], is the use of cellular automata in public-key cryptosystems. This system is still far too new and has not been studied extensively, but a preliminary examination suggests that it may have a cryptographic weakness similar to one seen in other cases [562]. Still, this is a promising area of research. Cellular automata have the property that, even if they are invertible, it is impossible to calculate the predecessor of an arbitrary state by reversing the rule for finding the successor. This sounds a whole lot like a trapdoor one-way function.

20.8 OTHER PUBLIC-KEY ALGORITHMS

Many other public-key algorithms have been proposed and broken over the years. The Matsumoto-Imai algorithm [1021] was broken in [450]. The Cade algorithm

was first proposed in 1985, broken in 1986 [774], and then strengthened in the same year [286]. In addition to these attacks, there are general attacks for decomposing polynomials over finite fields [605]. Any algorithm that gets its security from the composition of polynomials over a finite field should be looked upon with skepticism, if not outright suspicion.

The Yagisawa algorithm combines exponentiation mod p with arithmetic mod $p - 1$ [1623]; it was broken in [256]. Another public-key algorithm, Tsujii-Kurosawa-Itoh-Fujioka-Matsumoto [1548] is insecure [948]. A third system, Luccio-Mazzone [993], is insecure [717]. A signature scheme based on birational permutations [1425] was broken the day after it was presented [381]. Tatsuaki Okamoto has several signature schemes: one is provably as secure as the Discrete Logarithm Problem, and another is provably as secure as the Discrete Logarithm Problem *and* the Factoring Problem [1206]. Similar schemes are in [709].

Gustavus Simmons suggested J-algebras as a basis for public-key algorithms [1455,145]. This idea was abandoned after efficient methods for factoring polynomials were invented [951]. Special polynomial semigroups have also been studied [1619,962], but so far nothing has come of it. Harald Niederreiter proposed a public-key algorithm based on shift-register sequences [1166]. Another is based on Lyndon words [1476] and another on propositional calculus [817]. And a recent public-key algorithm gets its security from the matrix cover problem [82]. Tatsuaki Okamoto and Kazuo Ohta compare a number of digital signature schemes in [1212].

Prospects for creating radically new and different public-key cryptography algorithms seem dim. In 1988 Whitfield Diffie noted that most public-key algorithms are based on one of three hard problems [492,494]:

1. Knapsack: Given a set of unique numbers, find a subset whose sum is N.
2. Discrete logarithm: If p is a prime and g and M are integers, find x such that $g^x \equiv M \pmod{p}$.
3. Factoring: If N is the product of two primes, either
   a) factor N,
   b) given integers M and C, find d such that $M^d \equiv C \pmod{N}$,
   c) given integers e and C, find M such that $M^e \equiv C \pmod{N}$, or
   d) given an integer x, decide whether there exists an integer y such that $x \equiv y^2 \pmod{N}$.

According to Diffie [492,494], the Discrete Logarithm Problem was suggested by J. Gill, the Factoring Problem by Knuth, and the knapsack problem by Diffie himself.

This narrowness in the mathematical foundations of public-key cryptography is worrisome. A breakthrough in either the problem of factoring or of calculating discrete logarithms could render whole classes of public-key algorithms insecure. Diffie points out [492,494] that this risk is mitigated by two factors:

> 1. The operations on which public key cryptography currently depends—multiplying, exponentiating, and factoring—are all fundamental arithmetic phenom-

ena. They have been the subject of intense mathematical scrutiny for centuries and the increased attention that has resulted from their use in public key cryptosystems has on balance enhanced rather than diminished our confidence.

2. Our ability to carry out large arithmetic computations has grown steadily and now permits us to implement our systems with numbers sufficient in size to be vulnerable only to a dramatic breakthrough in factoring, logarithms, or root extraction.

As we have seen, not all public-key algorithms based on these problems are secure. The strength of any public-key algorithm depends on more than the computational complexity of the problem upon which it is based; a hard problem does not necessarily imply a strong algorithm. Adi Shamir listed three reasons why this is so [1415]:

1. Complexity theory usually deals with single isolated instances of a problem. A cryptanalyst often has a large collection of statistically related problems to solve—several ciphertexts encrypted with the same key.

2. The computational complexity of a problem is typically measured by its worst-case or average-case behavior. To be useful as a cipher, the problem must be hard to solve in almost all cases.

3. An arbitrarily difficult problem cannot necessarily be transformed into a cryptosystem, and it must be possible to insert trapdoor information into the problem so that a shortcut solution is possible with this information and only with this information.