The Digital Signature Algorithm (DSA)

Johannes Buchmann

December 15, 2001

Contents

Chapter 1

Introduction

The Digital Signature Algorithm (DSA) has been developed by the Accredited Standards Committee on Financial Services (ASC X9) as part of standard X9.30-1997: Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry. That standard consists of two parts. Part 1: The Digital Signature Algorithm (DSA) (Revised), and Part 2: The Secure Hash Algorithm (SHA-1). The DSA defines a technique for generating and validating digital signatures. This technique is supposed to provide data integrity and non-repudiation of the origin and content of a digital message.

In this report I describe the current knowledge concerning the security of the DSA. Chapter 2 describes the DSA. Chapter 3 discusses the security of the DSA. In Chapter 4 I discuss the SHA-1 hash function. Chapter 5 gives a final evaluation of the DSA.

Chapter 2

Description of DSA

2.1 The algorithm

The Digital Signature Algorithm (DSA) has been suggested and standardized by the National Institute of Standards and Technology (NIST) of the U.S (see [5], [17]). It is an efficient variant of the ElGamal signature scheme (see [3] Section 11.4). The ElGamal signature scheme has several draw backs which the DSA repairs.

1. Given todays security standards, an ElGamal Signature is of bit length at least 2048. A DSA signature is of bit length 320.
2. Signature verification in the ElGamal scheme requires three modular exponentiations with exponents of bit length at least 1024. Signature verification in the DSA requires only two modular exponentiations with exponents of bit length 160.

The idea for the reduction of the exponent size is taken from a signature scheme of Schnorr [12] which is patented. There is a discussion whether the Schnorr patent covers the DSA.

2.1.1 Key generation

A prime number q is chosen (see Section 3.2.5) with

$$2^{159} < q < 2^{160}.$$

That prime number q has bit length 160. A prime number p is chosen with the following properties:

- $2^{511+64j} < p < 2^{512+64j}$ for some $j \in \{0, 1, \ldots, 8\}$,

- the prime number q, which was chosen first, divides $p-1$.

The bit length of p is between 512 and 1024. It is a multiple of 64. Therefore, the binary expansion of p can be written sequence of 8 to 16 bitstrings of length 64. The condition $q \mid (p-1)$ implies that the group $(\mathbb{Z}/p\mathbb{Z})^*$ contains $q-1$ elements of order q. The primes p and q can be system parameters for all users.

An $x \in \{1, \ldots, p-1\}$ is chosen with $x^{(p-1)/q} \not\equiv 1 \bmod p$ and

$$g = x^{(p-1)/q} \bmod p$$

is computed. Then the residue class $g + p\mathbb{Z}$ has order q in the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^*$ of residues mod p. It generates the uniquely determined subgroup of order q of $(\mathbb{Z}/p\mathbb{Z})^*$. Finally, a random a (see Section 2.3) in the set $\{0, 1, 2, \ldots, q-1\}$ is chosen and

$$A = g^a \bmod p$$

is computed. The signer's public key is (p, q, g, A). The signer's private key is a. Note that the residue class $A + p\mathbb{Z}$ is an element of the subgroup generated by $g + p\mathbb{Z}$. The order of this subgroup is approximately 2^{160}. Computing the secret key a from A requires the solution of a discrete logarithm problem in this subgroup. We will discuss the difficulty of this discrete logarithm problem below.

2.1.2 Signature generation

The signer wants to sign the document $x \in \{0,1\}^*$. He uses the publicly known hash function (see Chapter 4)

$$\text{SHA-1} : \{0,1\}^* \to \{0,1\}^{160}.$$

He chooses a random number $k \in \{1, 2, \ldots, q-1\}$ (see section 2.3), computes

$$r = (g^k \bmod p) \bmod q, \tag{2.1}$$

and sets

$$s = k^{-1}(\text{SHA-1}(x) + ar) \bmod q. \tag{2.2}$$

Here, k^{-1} is the inverse of k modulo q. The signature of x is (r, s).

2.1.3 Verification

The verifier wants to verify the signature (r, s) of the document x. He obtains the signer's authentic public key (p, q, g, A). Then he verifies that

$$1 \le r \le q-1 \text{ and } 1 \le s \le q-1. \tag{2.3}$$

If this condition is violated, then the verifier rejects the signature. Otherwise, he verifies that

$$r = ((g^{(s^{-1}h(x)) \bmod q} A^{(rs^{-1}) \bmod q}) \bmod p) \bmod q. \tag{2.4}$$

If the signature is constructed according to (2.1) and (2.2), then (2.4) holds. In fact, the construction implies

$$g^{(s^{-1}h(x)) \bmod q} A^{(rs^{-1}) \bmod q} \equiv g^{s^{-1}(h(x)+ra)} \equiv g^k \bmod p,$$

which implies (2.4).

2.2 Prime number generation

We describe the prime number generation for the DSA public key. The algorithm for choosing the prime number q is as follows.

1. Choose Seed $\in \{0,1\}^*$, $g = |\text{Seed}| \geq 160$.
2. Compute $U = \text{SHA-1}(Seed \oplus \text{SHA-1}(\text{Seed} + 1) \bmod 2^g$.
3. Set q to the number that is obtained by setting the most and least significant bit of U to 1. Then $2^{159} < q < 2^{160}$.
4. Use a probabilistic primality test with error probability at most 2^{-80} to test whether q is prime. The Miller-Rabin test (see [3] Chapter 6) is mentioned as an example for such an algorithm.
5. If q is not prime, then go to step 1. Otherwise, output q.

Once q is found, the prime number p is constructed as follows.

1. Choose $j \in \{0,1,\ldots,8\}$. The prime p will satisfy $2^{L-1} < p < 2^L$ with $L = 512 + 64j$.
2. Divide $L-1$ with remainder by 160, that is, find integers n, b such that $L - 1 = 160n + b$, $0 \leq b < 160$.
3. Set counter $= 0$, offset $= 2$.
4. For $k = 0, 1, \ldots, n$ let

$$V_k = \text{SHA-1}\big((\text{Seed} + \text{offset} + k) \bmod 2^g\big).$$

5. Set

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

and

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

Then $0 \leq W < 2^{L-1}$ and

$$2^{L-1} \leq X < 2^{L}.$$

6. Set $c = X \bmod 2q$ and $p = X - (c - 1)$. Then $p \equiv 1 \bmod 2q$, that is, $2q$ divides $p - 1$.
7. If $p < 2^{L-1}$, then go to step 9.
8. Use a probabilistic primality test with error probability at most 2^{-80} to test whether p is prime. The Miller-Rabin test (see [3] Chapter 6) is mentioned as an example for such an algorithm. If p passes this test, then go to step 11.
9. Set counter to counter + 1 and set offset to offset + n + 1.
10. If counter $\geq 2^{12} = 4096$ generate a new prime number q and start the process of generating p again.
11. Save the value of Seed and the value of counter for use in certifying the proper generation of p and q. Those quantities have to be kept secret.

2.3 Random number generation

In the DSA, the secret key $a \in 1, \ldots, q - 1$ and for each signature an exponent $k \in \{1, \ldots, q - 1\}$ are pseudo-randomly generated. We describe that pseudo-random number generation.

The pseudo-random number generator uses a one-way function

$$G : \{0,1\}^{160} \times \{0,1\}^{b} \rightarrow \{0,1\}^{160} \tag{2.5}$$

where $160 \leq b \leq 512$. That function can be constructed using the SHA-1 hash function or the DES encryption algorithm. In the first case, the value b is chosen from $\{160, \ldots, 512\}$. In the second case, the value b is set to 160.

2.3.1 Generation of the secret key a

The generation of m secret keys a_j, $0 \leq j \leq m - 1$ works as follows.

1. Choose a secret XKEY $\in \{0,1\}^{b}$.
2. Let t be the hexadecimal number 67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0.

3. For $j = 0, 1, \ldots, m-1$ do the following:
   (a) Optionally choose $\text{XSEED}_j \in \{0,1\}^*$.
   (b) Set $\text{XVAL} = (\text{XKEY} + \text{XSEED}_j) \bmod 2^b$.
   (c) Set $a_j = G(t, \text{XVAL}) \bmod q$.
   (d) Set $\text{XKEY} = (1 + \text{XKEY} + a_j) \bmod 2^b$.

2.3.2 Generation of k

The precomputation of m values k, k^{-1}, r for m DSA-signatures works as follows.

1. Choose a secret $\text{KKEY} \in \{0,1\}^b$.
2. Let t be the hexadecimal number EFCDAB89 98BADCFE 10325476 C3D2E1F0 67452301.
3. For $j = 0, 1, \ldots, m-1$ do the following:
   (a) Set $k = G(t, \text{KKEY}) \bmod q$.
   (b) Set $k_j^{-1} = k^{-1} \bmod q$.
   (c) Set $r_j = (g^k \bmod p) \bmod q$.
   (d) Set $\text{KKEY} = (1 + Kkey + k) \bmod 2^b$.

After m signatures are calculated using those values for k and r the value t is set to the SHA value of the mth message.

Chapter 3

Security

In this section, we discuss the security of the the DSA. We first explain the state of the discussion concerning the security of signature schemes. Then we specifically discuss the security of the DSA.

3.1 Security requirements

3.1.1 Security proofs are reductions

The security of all known digital signature schemes depends on the intractability of certain computational problems in mathematics, specifically in number theory. Examples are the integer factoring problem and the discrete logarithm problem in an appropriate group. In the case of DSA, the group is a subgroup of prime order in the multiplicative group of a finite prime field.

However, no provably hard computational problems are known which can serve as the security basis of a digital signature scheme. Therefore, no rigorous security proofs for signature schemes are known and there is little hope that such proofs will be found in the future.

Todays security proofs are reductions. The goal of such a reduction is to show that the ability of an attacker to mount a successful attack on a signature scheme implies his ability of solving a basic computational problem in mathematics. This is supposed to increase the trust in the security of a digital signature system. The idea of this approach is the following.

When analyzing the security of a signature scheme it is hard to predict which attacks are possible since the system may be very complex and may depend on numerous parameters. Even, if the underlying basic computational problems are intractable, some part of the signature scheme might be implemented in such a way that an attack is possible. A famous example for such a situation is

the discovery of the possibility of an attack against the RSA encryption standard PKCS # 1 (see [2]). In this standard, an insecure padding scheme was used, which compromised the security of the whole scheme, even though the RSA encryption scheme is based on the intractable integer factoring problem. However, if the security of the digital signature scheme can be reduced to the difficulty of a well defined computational problem in mathematics, then, in order to evaluate the security of the digital signature scheme, it is sufficient to study the difficulty of the underlying problem. The difficulty of the underlying mathematical problem can be studied thoroughly and, therefore, the level of security of the signature scheme is easier to estimate.

Such a security reduction also solves another problem. It is possible that a weakness of a digital signature scheme is discovered, for example, by a government agency of some country. That agency may then try to keep this weakness secret and take advantage of it. However, if the weakness of the signature scheme implies that a basic computational problem is no longer intractable, then keeping this weakness secret may be more difficult, since the solution of important scientific problems can be expected to happen at the same time independently in different places. Hence, reduction proofs make it less likely that a security hole can be abused.

It is an important question what the computational problems are to which the security of digital signature schemes should be reduced in order for the scheme to be considered more secure. Clearly, breaking a digital signature scheme in one of the ways explained below, can be considered to be a computational problem. In this sense, the security of any digital signature scheme can be trivially reduced to the intractability of a computational problem, namely to the problem of breaking the scheme. However, evaluating the computational difficulty of this problem is very difficult since it is very complex and has many parameters. This is even more true since breaking a digital signature scheme is a so called *interactive problem*, that is, in that problem several parties are involved: a signer, who knows his secret key, an attacker who does not know that key but wants to generate valid signatures of the signer, and perhaps an honest verifier. In the process of forging signatures the attacker can try to use the help of the signer and the verifier(see Section 3.1.5).

To make the security level of a digital signature scheme easy to evaluate, it is desirable to reduce its security to easy to specify *non-interactive* computational problems. An example is the factoring problem for RSA-modules: Given an integer n which is the product of two large primes p and q, find those factors p and q. It would be optimal to reduce the security of a digital signature scheme to problems which are of mathematical interest independently of their cryptographic applications. Then, the difficulty of those problems would be studied also outside the crypto community and would therefore be easier to evaluate. However, digital signature schemes whose security can be reduced to such problems seem not to be known. In the known reductions, the computational problems depend to a certain extent on the specific digital signature

scheme whose security is reduced to them. In my opinion, the less the computational problems depend on the digital signature schemes the stronger the security proof by reduction is.

3.1.2 Security of the secret key

A minimum requirement for secure digital signature schemes is the security of the secret key. An attacker has access to the public key of the signer. In a secure digital signature scheme, the determination of the secret key from the public key must be infeasible.

3.1.3 Existential forgery

Suppose that the problem of computing the secret key from the public key is intractable. This does not necessarily mean that the digital signature scheme is secure. It may still be possible that an attacker is able to generate valid signatures without the knowledge of the secret key.

In an *existential forgery* the attacker produces such a signature. In such a forgery, the attacker is not required to have control over the document which is signed. The only requirement is, that the result of an existential forgery a new signature of some document which has been produced without the knowledge of the secret key.

3.1.4 No message attacks

A *no message attack* or a *passive attack* is an existential forgery in which the attacker only knows the public key of the signer and has no access to further information such as valid signatures of other documents. A digital signature scheme is considered to be secure against no message attacks, if the possibility of such an attack implies the ability of solving a computational problem which is considered to be intractable.

3.1.5 Adaptive chosen message attacks

I explain the strongest security notion known for digital signature schemes: the security against existential forgery using an *adaptive chosen message attack*.

In an *adaptive chosen message attack* the adversary knows the public signature key of the signer and obtains valid signatures of a sequence of messages of his choice. The messages in the sequence may depend on signatures of previous messages. The goal of the adversary is an *existential forgery*, i.e. he wants to produce a new signature which has not been generated by the legitimate signer. In particular, the signature is not in the sequence of messages whose signatures

the attacker has obtained. But the newly signed message is not necessarily a message of the attackers choice.

One practical application of this notion is as follows. Suppose that a signature scheme is used in a challenge response identification. Then the verifier generates challenges which the prover is supposed to sign, thereby proving his identity. Those challenges can be generated as a sequence of adaptive chosen messages. If an adaptive chosen message attack makes existential forgery possible, then the verifier is able to forge valid signatures without knowing the secret key. The use of signature schemes in challenge response identification is quite common.

I explain a method for proving security against chosen message attacks more precisely. In a chosen message attack the attacker can generate a sequence of pairs (message, signature). A message in that sequence may depend on the previous pairs. The signature generation algorithm is probabilistic. Therefore, the signatures are generated according to some probability distribution. The signature scheme is considered secure against a chosen message attack if it is secure against no message attacks and if without using the secret key it is possible to generate a sequence which is algorithmically indistinguishable (see [1]) from the sequence which is generated using the signature algorithm. The idea of this concept is the following. If an existential forgery is possible using an adaptively chosen sequence of pairs (message, signature), then an existential forgery is possible using the algorithmically indistinguishable simulation of such a sequence. This latter existential forgery is a no message attack since the signing algorithm is not used. However, the digital signature scheme is known to be secure against no message attacks. Therefore, an adaptive chosen message attack is impossible.

It is common belief that security against adaptive chosen message attacks is the strongest possible security notion for digital signature schemes. In other words, no attack against a digital signature scheme is known which cannot be modeled as an adaptive chosen message attack. The role of this security notion is somewhat similar to the role of the model of a Turing machine in the theory of computation. No computing device is known which cannot be modeled as a Turing machine. However, no proof is known that no stronger computing model exists. Likewise, no proof is known that the security against adaptive chosen message attacks is the strongest possible security notion.

In my opinion, if a signature scheme is proven secure against adaptive chosen message attacks, then it can be considered secure in the strongest sense. However, there are no such proofs but only reductions (see Section 3.1.1).

3.1.6 Random oracle model

Security proofs for digital signature schemes are difficult since a digital signature scheme consists of many components and their interaction may be complicated. An important ingredient of most signature schemes are cryptographically secure

hash functions. The hash functions map very long messages to short strings of fixed length. The security of hash functions is discussed in Section 4.1. There I explain that no provably secure cryptographic hash functions are known.

If the security of a signature scheme is analyzed in the random oracle model (see [9], [4]), then the concrete hash function which is used in the digital signature scheme, is replaced by a so called *random oracle*. A random oracle can be viewed as a black box which contains a random function which maps long strings to short strings of fixed length. Nothing is known about this function, but it can be evaluated by making an explicit query. A typical proof of security against passive attacks in the random oracle model works as follows. If it is possible to come up with a forged signature for a document using one random oracle then such a forgery is also possible with another random oracle, resulting in another falsified signature (forking lemma, see [9]). The two valid signatures of the same document can then be used to solve an underlying mathematical problem.

Does a security proof in the random oracle imply the security of the real digital signature scheme in which a concrete hash function is used? Such an implication cannot be proved today. However, assuming that the concrete hash function behaves like a random oracle, a security proof in the random oracle model makes the security of the real scheme more plausible. On the other hand, there exist insecure signature schemes that can be proved secure in the random oracle model (see [4]). Those schemes look fairly artificial. Nevertheless, their existence raises the question what security proofs in the random oracle model really prove.

In my opinion, security proofs in the random oracle cannot prove the security of digital signature schemes but they make their security more plausible.

3.2 Choice of the parameters

3.2.1 Choice of q

The secret DSA key is a discrete logarithm in the subgroup of $(\mathbb{Z}/p\mathbb{Z})^*$ of order q and so is the secret k in each signature. The most efficient conventional algorithms, that become more efficient as q becomes smaller, for solving this problems are the Shanks baby-step giantstep algorithm [13] and the Pollard rho algorithm [10] and their variants. The run time of those algorithms is of order $q^{1/2}$.

If quantum computers can be built, then any discrete logarithm problem in $(\mathbb{Z}/p\mathbb{Z})^*$ can be solved in polynomial time (see Shor [14]) and DSA is insecure.

Lenstra and Verheul [7], assuming technological and algorithmic progress, recommend secure sizes of q as a function of time. This is shown in Table 3.1.

In the DSA, the prime number q is of fixed bit length 160. Given the pre-

Year	Bit length of q
2002	127
2010	138
2020	151
2030	165
2040	179
2050	193

Table 3.1: Secure size of q

dictions of Lenstra and Verheul [7], the DSA will be insecure by 2030. However, it is possible to change the specification of the DSA accordingly. This means changing the size of the prime number q and the output size of SHA-1.

3.2.2 The function $L_x[u, v]$

The most efficient algorithm for solving the general DL problem in $(\mathbb{Z}/p\mathbb{Z})^*$ is the Number Field Sieve (see Section 3.2.3) To estimate the running time and storage requirement of the Number Field Sieve the function

$$L_x[u, v] = e^{v(\log x)^u (\log\log x)^{1-u}}$$

is used, where x, u, v are positive real numbers. I explain the meaning of this function. We have

$$L_x[0, v] = e^{v(\log x)^0 (\log\log x)^1} = (\log x)^v \tag{3.1}$$

and

$$L_x[1, v] = e^{v(\log x^1 (\log\log x)^0} = e^{v \log x}. \tag{3.2}$$

Let x be a positive integer which is the input for an algorithm. In the context of this evaluation, x is the prime number p.

If an algorithm has running time $L_x[0, v]$, then by (3.1) it is a polynomial time algorithm. Its complexity is bounded by a polynomial in the size of the input. The algorithm is considered efficient, although its real efficiency depends on the degree v of the polynomial.

If the algorithm has running time $L_x[1, v]$, then by (3.2) it is exponential. Its complexity is bounded by an exponential function in the length of the input. The algorithm is considered inefficient.

If the algorithm has running time $L_x[u, v]$ with $0 < u < 1$, then it is *subexponential*. The algorithm is slower than polynomial but faster than exponential. So the function $L_x[u, v]$ can be viewed as a linear interpolation between polynomial time and exponential time.

3.2.3 Choice of p

The secret DSA key is a discrete logarithm in the subgroup of $(\mathbb{Z}/p\mathbb{Z})^*$ of order q and so is the secret k in each signature. This problem can also be attacked by discrete logarithm algorithms in $(\mathbb{Z}/p\mathbb{Z})^*$. The fastest general discrete logarithm algorithm in $(\mathbb{Z}/p\mathbb{Z})^*$ is the Number Field Sieve (NFS) which has a conjectured running time of $L_q[1/3, C + o(1)]$ with $C = (64/9)^{1/3} = 1.9229...$ (see [11]). For special primes p, for example for $p = r^n + a$, where r and a are small and n is large, we have $C = 1.5262...$ or even less (SNFS).

The current record computation with the SNFS is the solution of the McCurley challenge problem [8] to compute the discrete logarithm modulo a prime of 129 decimal digits by Denny and Weber [16]. The prime involved was of a special form so that the special number field sieve could be applied.

The possibility of applying the special number field sieve must be avoided. In the DSA this is done by choosing the prime p randomly. Then the probability for p to be of a form that makes the Special Number Field Sieve applicable is negligible.

The current record for the general NFS is the computation of discrete logarithms modulo a 120 digits prime. This was done in 10 weeks, on a unique 525MHz quadri-processors Digital Alpha Server 8400 computer by Joux and Lercier [6].

Lenstra and Verheul [7] suggest field sizes for secure crypto systems as a function of time. This is shown in Table 3.2.

Year	Field size
2002	1028
2010	1369
2020	1881
2030	2493
2040	3214
2050	4047

Table 3.2: Secure cryptographic field sizes

Given those predictions, the current maximal field size in the DSA appears not to be sufficient. However, it is easily possible to change the specification of the DSA accordingly.

3.2.4 Choice of k

A new k has to be chosen for each signature. Otherwise, the secret key a can be determined by solving a linear system. We show how this works.

Suppose that the signatures s_1 and s_2 of the documents x_1 and x_2 are

generated with the same k. Then the number $r = (g^k \bmod p) \bmod q$ is the same for both signatures. Therefore,

$$s_1 - s_2 \equiv k^{-1}(\text{SHA-1}(x_1) - \text{SHA-1}(x_2)) \bmod q.$$

From this congruence, k can be determined if $h(x_1) - h(x_2)$ is invertible modulo q. From $k, s_1, r, \text{SHA-1}(x_1)$, the secret key a can be determined since

$$s_1 = k^{-1}(\text{SHA-1}(x_1) - ar) \bmod q$$

and therefore

$$a \equiv r^{-1}(\text{SHA-1}(x_1) - ks_1) \bmod q.$$

3.2.5 Prime number generation

The prime number generation in the DSA is described in Section 2.2. Basically, it works as follows. Pseudo-random 160-bit numbers are generated until a probable prime q is found. Next, pseudo-random L-bit numbers p are computed with $p \equiv 1 \bmod q$ until a probable prime is found. The pseudo-random number generator uses a secret seed and the SHA-1 hash function. The probabilistic primality test is required to have error probability at most 2^{-80}. The standard recommends the use of the Miller-Rabin test (see [3] Chapter 6). That test has provably the required property. The security of the pseudo-random number generator depends on the secrecy of the seed and the properties of SHA-1. In contrast to the primality test, no proof is known for the security of the pseudo-random number generator. In the next section we discuss the pseudo-random number generator.

3.2.6 Selection of the secret key a and k

The secret key a and the secret numbers k are generated using the pseudo-random number generators described in Section 2.3.

The pseudo-random number generator generates 160-Bit random numbers using the one-way function G. The function G can be constructed using DES or SHA-1. This appears to be cryptographically secure.

Those 160-bit numbers are used to generate uniformly distributed pseudo-random numbers in $\mathbb{Z}_q = \{0, ..., q-1\}$. In the method described in FIPS PUB 186-2, Appendix 3 this is not satisfied. The values in $\{0, ..., 2^{160} - q - 1\}$ are selected twice as often as the values in $\{2^{160} - q, ..., q - 1\}$. Daniel Bleichenbacher has found an attack which uses this weakness (see `http://www.lucent.com/press/0201/010205.bla.html`). His attack uses 2^22 known signatures. The details of the attack have not appeared yet. NIST has published the NIST-Change Notice (October 5 2001) (see `http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf`). There, the following modification is suggested. Two numbers $z_1 = G(t, \text{KVAL}_1)$ and $z_2 = G(t, \text{KVAL}_2)$ are

generated and the random exponent $k = (z_1 * 2^{160} + z_2)(mod q)$ is used. This yields a very good approximate equidistribution.

3.3 Security proofs

No security proofs are known for the DSA. However, in the random oracle model, the security of a modification of the ElGamal signature scheme, from which the DSA is derived, can be reduced to the difficulty of computing discrete logarithms in $(\mathbb{Z}/p\mathbb{Z})^*$ (see [9]). In this modification the hash value is replaced by the hash value $h = h(x||r)$ of the document x concatenated with the commitment r. Also, the commitment r is not reduced mod q and the verification congruence is

$$r^s \equiv g^h A^r \bmod p. \tag{3.3}$$

This proof may be seen as a weak indication of the security of the DSA.

3.4 Existential forgery

We discuss the possibility of an existential forgery. Suppose an attacker wants to generate a document x and a valid signature (r, s) on that document.

A first possibility is to use a signature (r, s) on a different document x' and to find an x such that SHA-1(x) = SHA-1(x'). Given the current knowledge concerning the hash function SHA-1 this appears to be infeasible.

A second possibility is to find x, h, r, s such that the verification equation (3.3) holds. If in the DSA the challenge r is not reduced mod q and the verification congruence (3.3) is used, then we have the following. For fixed x, r, h this is a discrete logarithm problem in $(\mathbb{Z}/p\mathbb{Z})^*$. For fixed h, r, s this is a discrete logarithm problem in $(\mathbb{Z}/p\mathbb{Z})^*$. Given current knowledge, those DL problems appear to be intractable.

However, it is unknown whether there is a method for simultaneously constructing x, h, r, s.

3.5 Security reductions

No security reductions for the DSA are known, not even in the random oracle model. However, Pointcheval and Stern [9] prove that an ElGamal-variant and the Schnorr signature scheme are secure in the random oracle model. The DSA is derived from the ElGamal and the Schnorr scheme. Therefore, there may be variants of the DSA for which there are security reductions. However, it is unclear, what this means for the security of DSA.

Chapter 4

SHA-1 hash function

4.1 Hash functions

In signature schemes, hash functions are used to map long documents to short bit strings of fixed length that are actually signed. A *collision* of a hash function is a pair of different documents which are mapped to the same hash value. A hash function is called *collision resistant* if finding a collision of that hash function is intractable.

In signature schemes that sign hash values, the used hash functions must be collision resistant. Otherwise, if a collision (d, d') is found then the two documents d and d' have the same signature. If an attacker is able to obtain a valid signature of d, then he has also a signature for d'. For example, if the signer signs d in a challenge-response authentication, then he has also signed the other document d', possibly without knowing it. Collision resistant hash function are one-way functions. This means, that computing an inverse image for a given image is intractable. Therefore, in many cases, the use of collision resistant hash functions prevents existential forgeries since even if it is possible to generate a valid signature for a hash value it is impossible to find a document with that hash value.

Using the birthday paradox (see [3]) a collision for a hash function whose image has n elements can be found with probability $> 1/2$ by computing approximately $\sqrt{n}$ hash values. In view of the predictions in [7], the image of the hash function should at least contain 2^{160} elements in order to prevent the possibility of a birthday attack.

The birthday attack can be prevented if a *keyed hash function* is used. This is a function which maps a bit string and a key from a predefined key space to a hash value of fixed length. In the signing process, a random key is generated. The signature algorithm signs the hash value of a document that is generated by the hash function which is parameterized by the chosen key. The key is part

of the signature. It is also used in the verification process. In order for digital signature algorithm to be secure, the keyed hash function must be a *universal one-way hash function* (UOWF). This means that given a hash value and a key it is intractable to find a document such that the value of the hash function parameterized by the given key is the given hash value. No provably universal one-way hash function is known. However, there are constructions that use compression functions such as SHA-1 (see [15]) and are assumed to have the universal one-way property.

4.2 SHA-1

The hash function used in the DSA is SHA-1 (see [15]). It hashes to 160-bit hash values. The only known attack is the birthday attack. Given the current algorithmic knowledge, SHA-1 will be collision resistant for the next 20 years. However, there is no proof for the security of the DSA. Unexpected attacks may be discovered in the future.

Chapter 5

Conclusion

The DSA is an efficient digital signature algorithm. Given current algorithmic knowledge, the DSA appears to be secure for now.

However, when using the DSA several security issues have to be considered.

1. The intractability of the discrete logarithm problem in the subgroup of order q of $(\mathbb{Z}/p\mathbb{Z})^*$, the collision resistance of the SHA-1 hash function, and the indistinguishability of the pseudo-random number generator are necessary conditions for the security of the DSA. However, it is unknown, whether those assumptions are valid. Unexpected mathematical breakthroughs or quantum computers may make this assumption false.

2. There is no security proof for DSA nor is there a security reduction in any of the security models that cryptographers in the theory community have agreed on. So even if the underlying discrete logarithm problem and the underlying hash function and pseudo-random number generator are cryptographically secure, the DSA itself may still become insecure.

3. Given current algorithmic knowledge, the size of the security parameters suggested in the DSA is not sufficient to guarantee the security of the DSA for the next 20 years. A larger prime number p should be chosen. For longer security periods, a larger prime number q should be selected. This implies changing the output length of the hash function .

I recommend using the DSA with larger primes in an environment where it can easily be replaced by another signature scheme, if necessary.

Bibliography

[1] Bellare, M., and Goldwasser, S. Lecture notes on cryptography. `www-cse.ucsd.edu/usres/mihir`.

[2] Bleichenbacher, D. Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs # 1. In *Advances in Cryptology - Crypto '98* (1998), pp. 1–12.

[3] Buchmann, J. *Introduction to Cryptography.* Springer-Verlag, New York, 2001.

[4] Canetti, R., Goldreich, O., and Halevi, S. The random oracle methodology revisited. In *30th ACM Symp. on Theory of Computing (STOC)* (1998), pp. 209–218.

[5] FIPS 186-2, Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-2, U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, Virginia, 2000.

[6] Joux, A., and Lercier, R. Discrete logarithms in GF(p). Announcement on the NMBRTHRY Mailing List, 17. April 2001.

[7] Lenstra, A., and E.R.Verheul. Selecting cryptographic key sizes, October 1999.

[8] McCurley, K. The discrete logarithm problem. In *Cryptography and computational number theory, Proc. Symp. Apll. Math* (1990), C. Pomerance, Ed., vol. 42 of *Amer. Math. Soc.*, pp. 49–74.

[9] Pointcheval, D., and Stern, J. Security arguments for digital signatures and blind signatures. *J. Cryptology 13* (2000), 361–396.

[10] Pollard, J. Kagaroos, monopoly and discrete logarithms. *J. Cryptology 13* (2000), 437–447.

[11] Schirokauer, O. Discrete logarithms and local units. *Phil. Trans. R. Soc. London A 345* (1993), 409–423.

[12] SCHNORR, C. Efficient signature generation by smart cards. In *Advances in Cryptology - CRYPTO 89* (1991), Lecture Notes in Computer Science, Springer - Verlag, pp. 161–174.

[13] SHANKS, D. Class number, a theory of factorization and genera. In *Proc. Symp. Pure Math. 20* (1971), AMS, Providence, R.I., pp. 415–440.

[14] SHOR, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Computing 26* (1997), 1484–1509.

[15] STANDARD, S. H. National Institute of Standars and Technology (NIST), FIPS Publication 180-1, April 1995.

[16] WEBER, D., AND DENNY, T. The solution of mccurleys discrete logarithm challenge. In *Advances in Cryptology - Crypto 98* (1998), vol. 1462 of *LNCS*, pp. 458–471.

[17] ANSI X9.30:1-1997, Public Key Cryptography for the Financial Services Industry: Part 1: The Digital Signature Algorithm (DSA). Available from the ANSI X9 Catalog, 1997.

Index