Digital Signature Schemes

Introduction

The conventional handwritten signature on a document is used to certify that the signer is responsible for the content of the document. The signature is physically a part of the document and while forgery is certainly possible, it is difficult to do so convincingly. Trying to mimic a handwritten signature in a digital medium leads to a difficulty since cut and paste operations can be used to create a perfect forgery. Thus, we need to have a way of signing messages digitally which is functionally equivalent to a physical signature, but which is at least as resistant to forgery as its physical counterpart.

Introduction

Schemes which provide this functionality are called ***Digital Signature Schemes***. A Digital Signature Scheme will have two components, a private *signing algorithm* which permits a user to securely sign a message and a public *verification algorithm* which permits anyone to verify that the signature is authentic. The signing algorithm needs to "bind" a signature to a message in such a way that the signature can not be pulled out and used to sign another document, or have the original message modified and the signature remain valid. For practical reasons it would be necessary for both algorithms to be relatively fast and if small computers such as smart cards are to be used, the algorithms can not be too computationally complex.

There are many Digital Signature Schemes which meet these conditions, but we shall only investigate a few of the most popular ones.

RSA Signatures

As we have previously noted, in order for Bob to sign a message m, he raises m to his private decryption exponent mod n. This is the signature algorithm. Anyone can verify this signature by raising m^d to Bob's public encryption exponent mod n. This is the verification algorithm. Application of the verification algorithm to a valid signature yields the message m. The verifier must know the message m in order to be sure that this is the message that Bob signed, so in this application Bob must send the ordered pair (m, m^d mod n). Some care must be taken in the construction of the message to be signed in this way. For instance, if m is the instruction to Bob's bank to issue a check to Alice, then if Alice intercepts the ordered pair, she can send the same pair to Bob's bank whenever she is a little low on cash. To prevent this kind of abuse, when it matters, messages should include dates and other such items which prevent the message from being reused.

RSA Scheme

Forgeries of Bob's signature are easy to construct. The requirement for a valid signature is that raising the second coordinate to Bob's public encryption exponent e gives the first coordinate. Frank the "forger" can take any number y, calculate $x = y^e \bmod n$ and send the pair (x,y). This will be verified as Bob signing the message x. Frank's problem is that he has no control over the "message" x, which will normally be just random nonsense. Without breaking the RSA cryptosytem, Frank has only a negligible chance of finding a meaningful message, let alone a desired message.

El-Gamal Signature Scheme

Unlike the RSA Signature scheme, which can be used as both a cryptosystem and a signature scheme, this signature scheme is designed specifically for signatures and is based on the discrete logarithm problem. As with the El-Gamal cryptosystem, computations are carried out in $\mathbf{Z}_p$, where p is a prime such that the discrete log problem is intractable in $\mathbf{Z}_p$. A generator α of $\mathbf{Z}_p^*$ is fixed, and each user selects a secret exponent a, and publishes the value $\beta = \alpha^a \bmod p$. If Alice wishes to sign a message m, she will first select a random secret integer k with gcd(k,p-1) = 1. She then computes $r = \alpha^k \bmod p$ and then computes $s = k^{-1}(m - ar) \bmod (p-1)$. The signature is the triple (m,r,s).

El-Gamal Signature Scheme

The verification algorithm compares $\beta^r r^s$ mod p and α^m mod p.

Noting that from the definition of s, we have m = sk + ar mod (p-1), we see that:

$$\alpha^m = \alpha^{sk+ar} = (\alpha^k)^s (\alpha^a)^r = r^s \beta^r \bmod p.$$

El-Gamal Signature Scheme

Now suppose that Frank wants to forge Alice's signature on a message m without knowing Alice's secret exponent a. He can pick r randomly (just as Alice does) and then has to find an s so that $\beta^r r^s = \alpha^m$ mod p. Rewritten, this amounts to solving $r^s = \beta^{-r}\alpha^m$ mod p. Which is the discrete logarithm problem. On the other hand, if he first selects a random s, then he must solve the congruence $\beta^r r^s = \alpha^m$ mod p for r. This is a problem for which no feasible solution is known and it does not seem to be related to any well studied problem such as the Discrete Log problem. There remains the possibility that Frank can choose r and s simultaneously to get a valid signature. No one has discovered a way to do this, but then again, no one has proved that it can't be done.

El-Gamal Signature Scheme

Unlike the RSA signature scheme, Frank can not forge Alice's signature on "random messages" by randomly picking r and s and calculating a message m so that (m,r,s) is a valid Alice signature [to do this would require solving the discrete log problem]. However, Frank can create valid Alice signatures by selecting r,s and m simultaneously. To do this, Frank picks two integers, i and j (less than p-1) such that gcd(j,p-1) = 1. Then Frank calculates:

$$r = \alpha^i \beta^j \bmod p$$
$$s = -rj^{-1} \bmod (p-1)$$
$$m = is \bmod (p-1)$$

Checking the verification algorithm, we see that:

$$\begin{aligned} \beta^r r^s &= \beta^r(\alpha^i \beta^j)^s \bmod p \\ &= \beta^r(\alpha^{is}\beta^{js}) \bmod p \\ &= \beta^r(\alpha^{is}\beta^{-r}) \bmod p \\ &= \alpha^m \bmod p. \end{aligned}$$

El-Gamal Signature Scheme

There are some protocol failures that would compromise the El-Gamal signature scheme. The first involves the secret exponent k. Should this become known, then given a signature (m,r,s) the congruence $ar = m-ks \bmod (p-1)$, has $d = \gcd(r,p-1)$ possible solutions for a. The correct one can be found by verifying that $\beta = \alpha^a \bmod p$. This gives Alice's secret exponent a and so breaks the system.

El-Gamal Signature Scheme

A second protocol failure would occur if Alice used the same exponent k for two different messages. If Alice did this for two messages, m_1 and m_2, then the r in the signatures would be the same (and Frank would notice this failure of protocol). If the corresponding s values are s_1 and s_2, then

$-ar = s_1k - m_1 = s_2k - m_2 \bmod (p-1)$, and so,

$(s_1 - s_2)k = m_1 - m_2 \bmod (p-1)$.

Let $d = \gcd(s_1-s_2, p-1)$. There are d solutions to the congruence, and d will usually be small. The correct value of k can be obtained by checking these d possibilities to see which satisfies $r = \alpha^k \bmod p$. Once k is determined, we can calculate a as above and break the system.

Hash Functions

One of the problems with the above mentioned signature schemes is that the signatures are as long or longer than the messages that they sign. When the messages are large this can become a significant difficulty. One way to deal with this is to use ***cryptographic hash functions***. A hash function h takes a message m of arbitrary length and produces a *message digest* h(m) of some fixed length. In order for a hash function to be useful in cryptographic work, it should satisfy the following conditions:

1. The message digest h(m) should be calculated very quickly.
2. The hash function h should be a one-way function, that is, given a message digest h(m), it should be computationally infeasible to obtain the message m.
3. The hash function h should be ***strongly collision free***, meaning that it should be computationally infeasible to find two messages m_1 and m_2 so that $h(m_1) = h(m_2)$.

Hash Functions

There are several professional strength hash functions available. In 1990, Rivest proposed the **MD4** Hash function and the following year he presented a strengthened version known as **MD5**. These hash functions produce message digests of 128-bit size for arbitrary length messages. In 1993 (and modified in 1994) the federal government adopted the **Secure Hash Standard (SHS)** which produces message digests of 160-bits. These functions (algorithms) are fast but complicated and their analysis is difficult. No proof is known that they do satisfy the 2nd and 3rd conditions for a cryptographic hash function, but in practice they seem to work quite well.

We shall examine the *discrete log hash function* due to Chaum, van Heijst and Pfitzmann. Unfortunately, the calculation of this function is too slow to be of practical use, but it is simple enough to permit an analysis of its cryptographic security.

Hash Functions

Select a large prime p such that $q = (p-1)/2$ is also prime. Choose two primitive roots α and β of $\mathbf{Z}_p$. For a message m with $m < q^2$ we write $m = b + cq$ with $0 \leq b,c < q$. We then set $h(m) = \alpha^b\beta^c \bmod p$. In this set up, since α is a primitive element, there exists an exponent a such that $\beta = \alpha^a \bmod p$. Of course finding this exponent involves solving the discrete log problem in this field. If, however, we can find a collision for this hash function, i.e., different messages $m_1 = r + sq$ and $m_2 = t + uq$ with $h(m_1) = h(m_2)$, then we can easily calculate the exponent a. This follows since,

$$\alpha^r\beta^s = \alpha^t\beta^u \bmod p, \text{ implies}$$
$$\alpha^{r-t} = \beta^{u-s} = \alpha^{a(u-s)} \bmod p, \text{ and so,}$$
$$a(u-s) = (r-t) \bmod (p-1).$$

Let $d = \gcd(u-s, p-1)$. There will be d solutions (for a) of this last congruence.

Hash Functions

Since $p - 1 = 2q$ and q is prime, the only possibilities for d are $1,2,q$ and $2q = p-1$. Since $0 \leq u,s < q$, we have that $-q < u-s < q$, so if u is not equal to s, any divisor of $u-s$ (such as d) must be less than q, i.e., $d = 1$ or 2. On the other hand, if $u = s$, then from the congruence we obtain $r = t$, so the messages are the same, contrary to our assumption. If
$d = 2$, we can check the two possible values to obtain the correct exponent a. So, if we assume that finding discrete logarithms is computationally infeasible, then it follows that this hash function is strongly collision free.

Hash Functions

In the digital signature application, a cryptographic hash function is made public. The signature algorithm is then applied to the message digest obtained from this hash function. The message and this signature are then sent. The verification algorithm is applied to the signed message digest and compared with the message digest that is recomputed from the message. Signatures in this version are much shorter than the messages that they sign. The cryptographic properties of the hash function prevent forgeries. However, since the message digests are of fixed size, there are not as many of them as there are possible messages. This leads to another type of attack which can be launched against digital signature schemes which employ hash functions.

Birthday Attacks

The well known "Birthday Paradox" (not really a paradox, just a surprising result) that in a randomly selected group of at least 23 people, the probability of two having the same birthday is at least 1/2, leads to a method for finding collisions of a hash function known as the ***Birthday Attack***.

Consider a set Z with n elements (think of this as the set of hash digests). We wish to calculate the probability that k randomly selected elements of Z will contain no equal elements (no collisions). As the probability of selecting a particular element is 1/n, we calculate this probability as follows: The first choice is arbitrary. The probability that the second choice is distinct from the first is 1-1/n, while the probability that the third is distinct from the first two is 1 - 2/n, etc.

Birthday Attacks

Thus, the probability that k elements are selected with no collisions is

$$\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\ \cdots\ \left(1-\frac{k-1}{n}\right) = \prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right).$$

If x is a small real number, then 1-x ≈ e^{-x} which is derived by taking the first two terms of the series expansion

$$e^{-x} = 1 - x + x^2/2! - x^3/3! \ldots .$$

Therefore, an estimate for our probability is

$$\prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right) \approx \prod_{i=1}^{k-1} e^{-\frac{i}{n}} = e^{-\frac{k(k-1)}{2n}}.$$

Birthday Attacks

Letting p be the probability of obtaining a collision, we have p ≈ 1 - $e^{-k(k-1)/2n}$. So,

$$e^{-\frac{k(k-1)}{2n}} \approx 1-p$$

$$\frac{-k(k-1)}{2n} \approx \ln(1-p)$$

$$\frac{k(k-1)}{2n} \approx \ln(\frac{1}{1-p})$$

$$k(k-1) \approx 2n\ln(\frac{1}{1-p})$$

$$k \approx \sqrt{2n\ln(\frac{1}{1-p})}$$

With p = ½, we have: $k \approx 1.17\sqrt{n}$.

Thus, by selecting just slightly over SQRT(n) random choices from Z, we obtain a collision with probability at least 50%.

Birthday Attacks

In the Birthday Paradox, $n = 365$ and our approximation gives $k \sim 22.3$. In the Birthday attack, if the message digests were of x-bit length, there would be $n = 2^x$ digests, and by selecting $2^{x/2}$ arbitrary messages and applying the hash function to them, there will be a 50% chance of obtaining a collision. Thus, for 40-bit message digests, just over 2^{20} (about a million) random messages would be needed to find a collision with 50% probability. This is not very secure. It is usually suggested that the minimum acceptable size of a message digest is 128-bits to avoid a Birthday attack. The 160-bit message digest of DSS is even more secure against this attack.

Digital Signature Standard

The Digital Signature Standard (DDS) is a modification of the El-Gamal Signature Scheme. First proposed in 1991, it was adopted as a federal standard in 1994. The modification gives a signature to a 160-bit message which is only 320 bits long. Thus, the algorithm has been designed to work with a hash function that produces 160 bit message digests (such as the SHS).

Digital Signature Standard

The user of this scheme, say Alice, first finds a prime q which is 160 bits long and then chooses a prime p so that $q|p-1$. The discrete log problem should be hard for this prime p. (The initial version of the scheme had p chosen as a 512 bit number, but later versions permitted the size of p to be larger, up to 1024 bits.). Now, Alice chooses a qth root of unity mod p, that is an α such that $\alpha^q = 1 \bmod p$ (this can be done by finding a primitive root mod p, say g, and calculating $\alpha = g^{(p-1)/q} \bmod p$.) Alice then chooses a secret exponent a, with $0 < a < q-1$, and calculates $\beta = \alpha^a \bmod p$. The values of p, q, α, and β are made public and the exponent a is kept secret.

Digital Signature Standard

To sign a message m, Alice first selects a random secret integer k, with $0 < k < q-1$. She then computes, $r = (\alpha^k \bmod p) \bmod q$ and $s = k^{-1}(m + ar) \bmod q$. Her signature is then (m,r,s). In order for Bob to verify this signature, he computes $u = s^{-1}m \bmod q$ and $v = s^{-1}r \bmod q$. He then computes $w = (\alpha^u\beta^v \bmod p) \bmod q$ and accepts the signature if and only if $w = r$.

To see why this works, from the definition of s it follows that:

$$sk = (m + ar) \bmod q, \text{ so}$$

$$k = s^{-1}m + s^{-1}ar = u + av \bmod q.$$

Thus, $\alpha^k = \alpha^{u + av} = \alpha^u\beta^v \bmod p$. Therefore, taking the mod q values, we have $r = w$.

Digital Signature Standard

As in the El-Gamal scheme, the exponent a must be kept secret, and the secret numbers k should never be used twice. DSS is considered to be stronger than El-Gamal, since in this scheme the secret number k is harder to obtain from r because of the reduction mod q. The verification step in DSS is also faster than the corresponding step in El-Gamal, since there are fewer modular exponentiations to perform, and this is an important practical consideration.

Euler Pseudoprimes

If n is an odd composite number and b is an integer with (n,b) = 1 such that:

$$\boldsymbol{b^{(n-1)/2} \equiv \left(\frac{b}{n}\right) \ \mathbf{mod}\ n}$$

then n is called an **Euler pseudoprime** to the base b.

Suppose we wish to find the number of bases for which 45 is an Euler pseudoprime.

We first note that the only bases that need be considered are {1,2,4,7,8,11,13,14,16,17,19,22,23,26,28,
29,31,32,34,37,38,41,43,44}
i.e., those integers less than 45 which are not divisible by 3 or 5.

Euler Pseudoprimes

$$b^{(n-1)/2} \equiv \left(\frac{b}{n}\right) \bmod n$$

The left hand side is b^{22}, a non-zero square, and the right hand side is either 0, +1 or -1. Raising each of the possible bases to the 22nd power mod 45, shows that the only possible values are 1,19,26 and 44 = ±1, ±19 each giving a value of +1. Now calculate the Jacobi symbol for these choices: Note that in this case we have -

$$\left(\frac{b}{45}\right) = \left(\frac{b}{3}\right)^2 \left(\frac{b}{5}\right) = \left(\frac{b}{5}\right)$$

Euler Pseudoprimes

So, we have

$$\left(\frac{1}{5}\right) = 1$$

$$\left(\frac{19}{5}\right) = \left(\frac{4}{5}\right) = 1$$

$$\left(\frac{26}{5}\right) = \left(\frac{1}{5}\right) = 1$$

$$\left(\frac{44}{5}\right) = \left(\frac{4}{5}\right) = 1$$

So, these 4 bases have 45 as an Euler pseudoprime.