Digital Signatures

November 30, 2014

This stuff is pretty much all in the textbook, sections 9.1- 9.3.

1 The Basic Idea

A digital signature system is a public-key version of message authentication. Suppose, for example, that you are concerned that the software you just purchased really was produced and tested by a reputable source, and has not been altered by the addition of malicious code. The manufacturer could attach at tag computed by MAC over the binary code, but this requires that you and the manufacturer share a secret key. You cannot transfer the software to a third party without exposing the key.

In a digital signature system, the manufacturer uses a secret key for signing the binary code, and provides a public key for verifying the signature. The system consists of and algorithm for generating public-private key pairs, and two additional algorithms for applying signatures and verifying them:

$$\textit{sign} : m, K_{\text{priv}} \mapsto \sigma_{K_{\text{priv}}}(m)$$

$$\textit{verify} : m, s, K_{\text{pub}} \mapsto \{\text{yes}, \text{no}\}.$$

The algorithms should have the following property: If $(K_{\text{priv}}, K_{\text{pub}})$ is a public-private key pair, then

$$\textit{verify}(m, \sigma_{K_{\text{priv}}}(m), K_{\text{pub}}) = \text{yes}.$$

However, it should be computationally infeasible for an adversary who does not know the private key K_{priv} to produce a pair (m, s) such that

$$\textit{verify}(m, s, K_{\text{pub}}) = \text{yes},$$

even if the attacker has seen a lot of other pairs (m', s') with $m' \neq m$ that satisfy this property. Such a pair (m, s) constitutes a *forgery* of the private keyholder's signature on m. Thus if the verification algorithm accepts the signature, everyone can be assured that the m originates with the signer and has not been tampered with. Such a scheme also provides a property called *non-repudiation*: In our example scenario, the software manufacturer should not be able to claim that it did not produce the binary source m, since everyone can verify the manufacturer's signature s.

2 RSA Signatures

The first scheme for digital signatures, and still one of the most widely used, is the RSA algorithm that we saw for public-key encryption. We have the same public-private parameters N, e, and d. The simple version is that the signer signs by 'encrypting' with his private key:

$$m, d, N \mapsto s = m^d \bmod N.$$

The verifier takes the pair (m, s), together with the public key e, N,computes $s^e \bmod N$ and outputs 'yes' if $s^e \bmod N = m$ and 'no' otherwise.

However, just as there are problems with the analogous 'textbook' version of RSA encryption, this simple account of RSA signing has some flaws. For one thing, if the message to be signed is very long (as indeed it will be if we are signing something like binary source of a large computer program), then we would need to break it into many blocks and sign each block separately. This results in a very large file, twice as big as the original program, and it will require a lot of time to verify the signature.

For another thing, this makes it easy to forge a signature, knowing just the signer's public key (e, N). Take any value $0 < s < N$, and set $m = s^e \bmod N$. Then (m, s) is a valid signature. The message m is itself gibberish, but this is still a forgery. A variant of this attack that produces a forged signature on a meaningful message m is the following: Choose $0 < r < N$ a random value relatively prime to N, and compute $u = r^e m \bmod N$. If you can somehow obtain the signer's signature on u, you will have

$$u^d \bmod N = (r^e m)^d \bmod N = rm^d \bmod N,$$

and multiplying by $r^{-1} \bmod N$ gives $m^d \bmod N$, which is a valid signature on m. Again, this seems a bit far-fetched, since you will have to persuade the signer to sign the gibberish message u. Still, the existence of such attacks means that forgeries are possible, and this is precisely what we want to rule out.

For both these reasons, the real RSA signature algorithm is not quite as described above. It begins the same way, generating a modulus and a pair of exponents as for RSA encryption. But the signature is not applied directly to the message, but to a hash of the message, using apublicly known cryptographically secure hash function H whose output is smaller than the modulus N. The signature on m is then $s = H(m)^d \bmod N$. To verify the signature, we compute $s^e \bmod N$ and $H(m)$ and accept if these two are equal. This modification means that we only have to sign short strings, and it also rules out simple forgeries of the kind described above. (For example, we would have to find an m that hashes to $s^e \bmod N$, which presumably cannot be done for preimage-resistant hash functions.)

In practice, digital signatures are *always* applied to hashes of the messages rather than the messages themselves. This is one of the reasons that collision-resistance is such an important property for hash functions: If we could find a pair of meaningful messages m, m' that collided ($H(m) = H(m')$) and obtain a valid signature on m, then we would have a valid signature on m'.

3 El Gamal Signature Scheme

Here is an alternative signature scheme whose security is based on the conjectured hardness of the discrete log problem. The algorithm as describe here (and in the textbook) is not widely used *per se,* but variants of it, known as the *Digital Signature Algorithm* and the *Elliptic Curve Digital Signature Algorithm* are commonly used standards.

3.1 Key generation

Alice (the signer) chooses a large prime p, a primitive root $g \bmod p$, and a secret value $1 \leq a \leq p-2$. Her signing key is a. Her public key is $(g, p, g^a \bmod p)$.

3.2 Signing

1. Message $= m$. We use a hash function h such that hash lengths are less than the number of bits of p.
2. Choose random k relatively prime to $p-1$.
3. Set $r = g^k \bmod p$.
4. Set $s = ((k^{-1} \bmod p) \cdot (h(m) - ar)) \bmod p - 1$.
5. The signature is (m, r, s).

3.3 Verification

1. Compute $v_1 = (g^a)^r r^s \bmod p$ and $v_2 = g^{\cdot h(m)} \bmod p$.
2. Accept if $v_1 = v_2$.

3.4 Why does it work?

If the message is correctly signed,

$$\begin{aligned} r^s &= g^{(k \cdot k^{-1}) \cdot (h(m) - ar) \bmod p-1} \bmod p \\ &= g^{h(m)-ar} \bmod p \\ &= g^{h(m)} g^{-ar} \bmod p. \end{aligned}$$

Thus

$$\begin{aligned} v_1 &= g^{ar} \cdot g^{h(m)} g^{-ar} \bmod p \\ &= g^{h(m)} \bmod p \\ &= v_2. \end{aligned}$$

3.5 Example with artificially small parameters.

Let $p = 107 = 2 \cdot 53 + 1$. To find a primitive element, we look for a value such that $g^{53} \bmod 107 \neq 1$. The first thing we might try is $g = 2$, and that works. Let's choose $a = 46$ as the secret key. We then have

$$2^{46} \bmod 107 = 56,$$

so the public key used by signers is the triple

$$(2, 107, 56).$$

Alice wants to sign the message $m = 90$. With these small parameters, it really doesn't make sense to talk about a hash function, so we will just use $h(m) = m = 90$. Alice generates a one-time random secret $k = 77$, which satisfies the requirement of being relatively prime to $p - 1 = 106$. She then computes

$$77^{-1} \bmod 106 = 95.$$

$$r = 2^{77} \bmod 107 = 63.$$

$$m - ar = 90 - 46 \cdot 63 \cdot = -2808.$$

$$-2808 \bmod 106 = 54$$

$$s = 95 \cdot 54 \bmod 106 = 42.$$

The signature thus has the three components $(m, r, s) = (90, 63, 42)$.

The verifier receiving the signed message computes:

$$(g^a)^r \bmod p = 56^{63} \bmod 107 = 36.$$

$$r^s \bmod p = 63^{42} \bmod 107 = 105.$$

$$v_1 = (36 \cdot 105) \bmod 107 = 35.$$

$$v_2 = 2^{90} \bmod 107 = 35.$$

Since $v_1 = v_2$, the signature is accepted.

3.6 Reuse of the random value k.

The random value k used to generate the signature must not be used to sign two different messages. If it is, then this will be detected by a repetition of the value of r in signatures on two different messages. If it is, then we can do a little math to obtain the secret signing key. Let us suppose that after computing the signature in the example above, Alice uses the same key to sign $m' = 21$. We then have

$$\begin{aligned} s' &= k^{-1} \cdot (m' - ar) \bmod p - 1 \\ &= 95 \cdot (21 - 46 \cdot 63) \bmod 106 \\ &= 59 \end{aligned}$$

To de-clutter the notation, we will perform all the calculations below (additions, multiplications, and inverses) modulo 106. From the previous signature, we have $s = 42$, so

$$s = 42 = 42 \cdot 59^{-1} \cdot 59 = 60 \cdot 59 = 60s'.$$

The forger does not know the value of k, but can still use the result of multiplying the above equation by k and obtain

$$m - ar = 60(m' - ar),$$

so

$$59ar = 60m' - m.$$

Since 59 is invertible mod 106, and since we know the value of r, we can solve the equation above and find a.

$$a = (60 \cdot 21 - 90) \cdot (59 \cdot 63)^{-1} = (60 \cdot 21 - 90) \cdot 91 = 46.$$

This completely breaks the system, and as a result the attacker is able to forge signatures on any document. It is possible that we will encounter situations where the coefficient of ar in the equation we obtain this way is not invertible mod $p - 1$. However, if we use primes of the form $2q + 1$ where q is prime, we will find at most two possible solutions to the equation, and we can check them both.

3.7 Security.

If an attacker is able to calculate discrete logs modulo p at base g, then he can recover the private key a and forge signatures on any document. So the security of this method depends on the conjectured hardness of the discrete log problem. We don't know anything about the converse—it is conceivable that someone can break this system and forge signatures without being able to find these discrete logs.