26 AUTHENTICATION OF USERS AND PROGRAM CODES USING PASSWORDS AND DIGITAL CERTIFICATES

Now that we can do public-key encryption and digital signature, we need some mechanism to bind users to keys. The approach proposed by Diffie and Hellman when they invented digital signatures was to have a directory of the public keys of a system's authorized users, like a phone book. A more common solution, due to Loren Kohnfelder, is for a *certification authority* (CA) to sign the users' public encryption and/or signature verification keys giving certificates that contain the user's name, attributed such as authorizations, and public keys. The CA might be run by the local system administrator; or it might be a third party service such as Verisign whose business is to sign public keys after doing some due diligence about whether they belong to the principals named in them.

A certificate might be described symbolically as

$$C_A = Sig_{K_S}(T_S, L, A, K_A, V_A)$$

where (using the same notation as with Kerberos) T_S is the certificate's starting date and time, L is the length of time for which it is valid, A is the user's name, K_A is her public encryption key, and V_A is her public signature verification key. In this way, only the administrator's public signature verification key needs to be communicated to all principals in a trustworthy manner.

Certification is hard, for a whole lot of reasons. I'll discuss different aspects later — naming in Chapter 6, 'Distributed Systems', public-key infrastructures in Chapter 21, 'Network Attack and Defense', and the policy aspects in Part III. Here I'll merely point out that the protocol design aspects are much harder than they look.

Certification is hard, for a whole lot of reasons. I'll discuss different aspects later — naming in Chapter 6, 'Distributed Systems', public-key infrastructures in Chapter 21, 'Network Attack and Defense', and the policy aspects in Part III. Here I'll merely point out that the protocol design aspects are much harder than they look.

One of the first proposed public-key protocols was due to Dorothy Denning and Giovanni Sacco, who in 1982 proposed that two users, say Alice and Bob, set up a shared key K_{AB} as follows. When Alice first wants to communicate with Bob, she goes to the certification authority and gets current copies of public key certificates for herself and Bob. She then makes up a key packet containing a timestamp T_A, a session key K_{AB} and a signature, which she computes on these items using her private signing key. She then encrypts this whole bundle under Bob's public key and ships it off to him. Symbolically,

$$A \rightarrow B : C_A, C_B, \{T_A, K_{AB}, Sig_{K_A}(T_A, K_{AB})\}_{K_B}$$

In 1994, Martín Abadi and Roger Needham pointed out that this protocol is fatally flawed [2]. Bob, on receiving this message, can masquerade as Alice for as long as Alice's timestamp T_A remains valid! To see how, suppose that Bob wants to masquerade as Alice to Charlie. He goes to Sam and gets a fresh certificate C_C for Charlie, and then strips off the outer encryption $\{\ldots\}_{K_B}$ from message 3 in the above protocol. He now re-encrypts the signed key packet $T_A, K_{AB}, Sig_{K_A}(T_A, K_{AB})$ with Charlie's public key — which he gets from C_C — and makes up a bogus message 3:

$$B \rightarrow C : C_A, C_C, \{T_A, K_{AB}, Sig_{K_A}(T_A, K_{AB})\}_{K_C}$$

It is quite alarming that such a simple protocol — essentially, a one line program — should have such a serious flaw remain undetected for so long. With a normal program of only a few lines of code, you might expect to find a bug in it by looking at it for a minute or two. In fact, public key protocols are if anything harder to design than protocols that use shared-key encryption, as they are prone to subtle and pernicious middleperson attacks. This further motivates the use of formal methods to prove that protocols are correct.

Often, the participants' names aren't the most important things the authentication mechanism has to establish. In the STU-III secure telephone used by the US government and defense contractors, there is a protocol for establishing transient keys with forward and backward security; to exclude middleperson attacks, users have a *crypto ignition key*, a portable electronic device that they plug into the phone to identify not just their names, but their security clearance level. In general, textbooks tend to talk about identification as the main goal of authentication and key management protocols; but in real life, it's usually authorization that matters. This is more complex, as it starts to introduce assumptions about the application into the protocol design. (In fact, the NSA security manual emphasises the importance of always knowing whether there is an uncleared person in the room. The STU-III design is a natural way of extending this to electronic communications.)

One serious weakness of relying on public-key certificates is the difficulty of getting users to understand all their implications and manage them properly, especially where they are not an exact reimplementation of a familiar manual control system