67

Authentication and Delegation with Smart-cards

M. Abadi, M. Burrows, C. Kaufman, B. Lampson

October 22, 1990, Revised July 30, 1992

Systems Research Center

DEC's business and technology objectives require a strong research program. The Systems Research Center (SRC) and three other research laboratories are committed to filling that need.

SRC began recruiting its first research scientists in 1984—their charter, to advance the state of knowledge in all aspects of computer systems research. Our current work includes exploring high-performance personal computing, distributed computing, programming environments, system modelling techniques, specification technology, and tightly-coupled multiprocessors.

Our approach to both hardware and software research is to create and use real systems so that we can investigate their properties fully. Complex systems cannot be evaluated solely in the abstract. Based on this belief, our strategy is to demonstrate the technical and practical feasibility of our ideas by building prototypes and using them as daily tools. The experience we gain is useful in the short term in enabling us to refine our designs, and invaluable in the long term in helping us to advance the state of knowledge about those systems. Most of the major advances in information systems have come through this strategy, including time-sharing, the ArpaNet, and distributed personal computing.

SRC also performs work of a more mathematical flavor which complements our systems research. Some of this work is in established fields of theoretical computer science, such as the analysis of algorithms, computational geometry, and logics of programming. The rest of this work explores new ground motivated by problems that arise in our systems research.

DEC has a strong commitment to communicating the results and experience gained through pursuing these activities. The Company values the improved understanding that comes with exposing and testing our ideas within the research community. SRC will therefore report results in conferences, in professional journals, and in our research report series. We will seek users for our prototype systems among those with whom we have common research interests, and we will encourage collaboration with university researchers.

Robert W. Taylor, Director

Authentication and Delegation with Smart-cards

M. Abadi, M. Burrows, C. Kaufman, B. Lampson

October 22, 1990, Revised July 30, 1992

C. Kaufman works at Telecommunications and Networks, Digital Equipment Corporation, 550 King Street, Littleton, MA 01460.

Authors' Abstract

The authentication of users in distributed systems poses special problems because users lack the ability to encrypt and decrypt. The same problems arise when users wish to delegate some of their authority to nodes, after mutual authentication.

In most systems today, the user is forced to trust the node he wants to use. In a more satisfactory design, the user carries a smart-card with sufficient computing power to assist him; the card provides encryption and decryption capabilities for authentication and delegation.

Authentication is relatively straightforward with a sufficiently powerful smart-card. However, for practical reasons, protocols that place few demands on smart-cards should be considered. These protocols are subtle, as they rely on fairly complex trust relations between the principals in the system (users, hosts, services). In this paper, we discuss a range of public-key smart-card protocols, and analyze their assumptions and the guarantees they offer.

Contents

1 Introduction

In a secure distributed environment, there is a need for users to prove their identities to nodes, from mainframes to automatic teller machines. There is also a need, though less recognized, for nodes to prove their identities to users, as each user may trust different nodes to different extents. Furthermore, users must be able to delegate some of their authority to the nodes that they trust.

Authentication protocols serve for these purposes, typically relying on secrets and encryption (e.g., [2, 10, 11]). The authentication of users poses special problems, because users lack the ability to encrypt and decrypt.

In the simplest approach to user authentication, the user owns a secret (his password) that he gives to a node that he wishes to use. It is likely that the password will be short and memorable, or that the user will need to write it down. In either case, it may be easy for an attacker to discover the user's password and hence obtain all his rights.

This weakness can be eliminated by introducing a simple card with a small amount of read-only memory. Each user carries a card, and each card contains a different secret, which the node verifies before granting access to the user. The secret can be quite long, and hence hard to guess, but theft of the card is a significant danger.

A further improvement consists in introducing a Personal Identification Number (PIN), which the user types when he presents his card. Thus, theft of the card alone no longer suffices for a security breach. This scheme is essentially that used by most Automated Teller Machines (ATMs).

All of these approaches suffer from a serious flaw: the node must be completely trusted, as it obtains all of the user's secrets. It is impossible for a user to delegate only part of his authority, or to delegate his authority for only a limited time. Moreover, a malicious node could remember the user's secrets for future mischief. This threat seems important—consider, for example, the users of public terminals in hotels.

A smart-card with sufficient computing power solves these problems. The smart-card we envision has its own keyboard, display, clock, logic for performing public-key encryption [5, 12], and can be electrically coupled to the node.

Authentication would be relatively straightforward with this powerful smart-card. The smart-card might operate only when fed a PIN, so an attacker would need to steal the smart-card and to discover the user's PIN in order to impersonate the user. The keyboard and display allow the user to communicate directly with the smart-card, so that the node never sees the password. Since the smart-card performs public-key encryption, no secrets ever need be revealed to the node.

After authentication, the smart-card can sign timestamped certificates to delegate part of the user's authority for a limited time. For example, the smart-card could issue a certificate that allows the node to manipulate the user's remote files for the next hour. The timestamp provides protection against replay attacks, and careful use of lifetimes can prevent mischief by the node at a later time.

Unfortunately, no one is currently selling a smart-card of the type we have described, though one may be available at some time in the future. More realistic protocols that place weaker demands on smart-cards should therefore be considered. Various compatible solutions can be implemented with reduced degrees of security and user convenience. The protocols are subtler, as they rely on more complex trust relations between the principals (users, hosts, services) in the system.

Consider, for example, a smart-card with no clock. Such a smart-card is attractive because it avoids the need for a battery. Unfortunately, the smart-card can no longer generate timestamps for its certificates, and it cannot check that other certificates have not expired. A variety of replay attacks becomes possible, unless the card can obtain the time somehow. To counter replay attacks effectively, the card may obtain the time from a network time service, which must then be secure and trusted.

In this paper, we describe a range of public-key smart-card authentication protocols, with different compromises between cost and security. The protocols were developed in the context of the Digital Distributed System Security Architecture [6]. Most previous work focuses on user authentication using shared-key cryptography, with little discussion of delegation (see, for example, [3]). We believe that public-key cryptography is more suitable than shared-key cryptography for authentication and delegation, and that it is not prohibitively expensive when used wisely. The protocols considered can be based on RSA encryption [12], but other algorithms (e.g., [7]) could also be used.

We analyze the protocols with a logic of authentication. This is essentially the logic of Burrows, Abadi, and Needham [1], with a simple extension to handle secure and timely channels. It should be noted that our logical account is not the only one possible. However, the formalism enables us to describe the assumptions and the guarantees of each protocol, clarifying the trust relations between principals. Moreover, a logical account helps in avoiding certain security flaws commonly present in authentication protocols [1].

In the next section, we summarize the notation of the logic; the logic is discussed further in an Appendix. In later sections, we describe smart-card authentication protocols, starting with those that require the more ambitious smart-card designs and the weaker trust relations. We analyze three cases in some detail; the reader can interpolate between these. The informal descriptions (without the logic) are self-contained, and the formal passages may be skipped in a first reading.

2 Notation

In the analysis of smart-card protocols, we apply a logic for describing the beliefs of the principals in the course of authentication. Several sorts of objects appear in the logic: principals, encryption keys, formulas, and communication links. (The original logic does not discuss links.) In what follows, the symbols P, Q, and R range over principals; X and Y range over formulas; K ranges over encryption keys; L ranges over communication links. The constructs that we use are listed below.

P **believes** X

P **controls** X: P has *jurisdiction* over X. The principal P is an authority on X and should be trusted on this matter. For example, a certification authority may be trusted to provide the public keys of principals.

P **said** X: P *once said* X. The principal P sent a message including X, either long ago or during the current run of the protocol. In any case, P believed X when he sent the message.

P **said**$_L$ X: P *once said* X on the communication link L.

P **sees** X: P *sees* X. A message containing X has been sent to P, who can read and repeat X (possibly after doing some decryption).

P **sees**$_L$ X: P *sees* X on the communication link L.

fresh(X): X is *fresh*, that is, X has not been sent in a message at any time before the current run of the protocol. This usually holds for timestamps and for *nonces*—expressions invented for the purpose of being fresh.

timely(L): The link L is a *timely channel*: all messages on link L are known to have been sent recently.

$\overset{K}{\mapsto} P$: P has K as a *public key*. The matching *secret key* (the inverse of K, denoted K^{-1}) will never be discovered by any principal except P, or a principal trusted by P.

$\overset{L}{\prec} P$: The link L is a *secure channel* from P: all messages on link L are known to have been sent by P, or a principal trusted by P.

$P \overset{X}{\rightleftharpoons} Q$: X is a *secret* that will never be known to anyone but P and Q, and possibly to principals trusted by them. Only P and Q may use X to prove their identities to one another. An example of a shared secret is a password or a PIN.

$\{X\}_K$: This represents the formula X encrypted under the key K.

$\langle X \rangle_Y$: This represents X combined with the formula Y; it is intended that Y be a secret, and that its presence prove the identity of whoever utters $\langle X \rangle_Y$. In implementations, X may simply be concatenated with Y; our notation highlights that Y plays a special rôle, as proof of origin for X.

3 A protocol for ideal smart-cards

We start with a description of how user authentication would work given the ultimate smart-card, with its own keyboard, display, clock, and logic for performing public-key encryption. Other schemes are best thought of in terms of their differences from this scheme.

We assume that a distributed name service is available. The name service contains *certificates* with information about which nodes each user should trust and which public keys belong to which agents. A certificate is a statement that has been signed by a principal. Often, certificates are signed by a widely trusted service, known as a *certification authority*. A certificate typically includes a timestamp, with its time of issue, and sometimes an explicit lifetime, which limits its validity. The certification authority need not be on-line; the name service need not be trusted.

We also assume that principals know the time accurately enough to check the validity of certificates and to detect the replay of timestamped messages. A trusted time service is one way of achieving this.

We first describe the protocol informally, and then more formally.

3.1 Discussion

For the sake of concreteness, let us imagine that a user wants to prove his identity to a workstation. Moreover, the user wishes to allow the workstation to access files on his behalf.

1. The user sits down at the workstation and presents his smart-card. The smart-card is willing to give certain information, including the name of the user, to anyone.

2. The workstation now authenticates itself to the smart-card. Given the name of the user, the workstation can retrieve all the certificates the smart-card needs to determine that it is allowed to delegate authority to the workstation; it forwards these to the smart-card. The workstation also generates a new public key and a matching secret key; we refer to these keys as *delegation keys*. It sends the public key and a timestamp to the smart-card, signed with its own secret key.

3. The smart-card examines the information presented by the workstation, verifying the signatures and lifetimes on all of the certificates. At this point, the smart-card knows the public key of the workstation and that the workstation can be trusted to act on behalf of the user. The smart-card shows that it is satisfied on its display and requests the user's PIN. In addition, it might give the identity of the workstation, or at least an indication, such as a group nickname. It might also provide some means to identify the workstation's display, such as a name or location.

4. Now the user has some evidence of the workstation's identity. The user responds by entering a PIN into the keyboard of the smart-card, thus authorizing the use of this workstation.

5. The entry of the correct PIN indicates that the genuine user is present, rather than some smart-card thief. Hence, the card constructs and signs a *delegation certificate*. This delegation certificate authorizes anyone who can demonstrate knowledge of the secret delegation key to act on behalf of the user for a limited period of time. It sends that certificate to the workstation.

6. The workstation verifies that the delegation certificate is signed with the card's secret key and contains the public delegation key. At this point, the workstation has authenticated the user, in the sense that it knows that whoever is at the keyboard has a particular public key. The workstation has enough information to consult an access control list and to determine whether it should provide its services to this user. Moreover, the delegation certificate enables the workstation to convince another node, such as a file server, that the workstation acts on behalf of the user.

This completes the authentication process. The workstation now knows the user's identity. It can prove that it acts on the user's behalf by presenting the delegation certificate and proving that it knows the secret delegation key. The user also knows something about the workstation—perhaps its name, but at least that the workstation can be trusted, according to the certification authority.

For authentication, it does not suffice for the user and the workstation to know each other's names. They must also know that they are communicating via a particular keyboard and a particular display, and for example that no malicious principal is interposed between the user and the workstation. An implementation of this protocol should provide this guarantee.

An important variation on this scheme has to do with rôles. A user may need and want different privileges when he acts as member of a research group and as manager, for example. A user may also want different privileges depending on his trust of the systems that act on his behalf. Software and hardware trusted in one rôle should be prevented from gaining privileges reserved for another rôle. One way of providing this capability is to issue multiple smart-cards to the user, one for each rôle. A more satisfactory solution is to have a single smart-card support multiple rôles. By typing at the card's keyboard, the user could select a rôle, to be mentioned in the delegation certificate. In addition, the smart-card could restrict the rôles available, to save the user from misplaced trust in unsafe environments.

For the purposes of a logical analysis, it would be adequate to conceive of the user in each of his rôles as a different user. For simplicity, we do not discuss rôles further in this paper.

3.2 Notation and assumptions

The notation used in the remainder of this section is as follows:

- S is the certification authority, W the workstation, C the smart-card, U the user, and F a file server (or any other node) that the workstation contacts on behalf of the user;
- K_s, K_w, and K_c are the public keys of S, W, and C, respectively;
- K_d is the public delegation key, generated by W;
- PIN is U's personal identification number;
- T_w, T_c, T_s, T'_s, and T''_s are timestamps;
- I_c is the smart-card's keyboard (input to the card);
- O_c is the smart-card's display (output from the card);
- I_w is the workstation's keyboard (input to the workstation); and
- O_w is the workstation's display (output from the workstation).

The name C is useful only in connecting U with K_c, and need not be present in an implementation. Similarly, the names for I_w and O_w will quite likely be related to one another, and to W. For example, if I_w and O_w are referred to by location, then the names could be identical.

In order to analyze the protocol with the logic, we have to state its assumptions and describe it more formally. We leave the description of the protocol to the next subsection, and now proceed to discuss its assumptions. The assumptions naturally fall into several classes:

Assumptions about timestamps: the smart-card, the workstation, and the file server believe that certain timestamps were generated recently:

1. C **believes fresh**(T_s), C **believes fresh**(T'_s), C **believes fresh**(T_w);
2. W **believes fresh**(T_c), W **believes fresh**(T''_s);
3. F **believes fresh**(T_c), F **believes fresh**(T''_s).

Assumptions about keys and secrets:

1. S **believes** $\overset{K_c}{\mapsto} C$, S **believes** $\overset{K_w}{\mapsto} W$: the certification authority believes the smart-card's public key is K_c, and the workstation's public key is K_w;

2. C **believes** $\overset{K_s}{\mapsto} S$: the smart-card believes the certification authority's public key is K_s;

3. C **believes** $C \overset{PIN}{\rightleftharpoons} U$: the smart-card believes that PIN is a secret shared with the user;

4. W **believes** $\overset{K_s}{\mapsto} S$: the workstation believes the certification authority's public key is K_s;

5. W **believes** $\overset{K_d}{\mapsto} U$: the workstation believes that K_d is a good key for U (probably because W has constructed the key);

6. F **believes** $\overset{K_s}{\mapsto} S$: the file server believes the certification authority's public key is K_s.

Assumptions about channels:

1. W **believes** $\overset{O_w}{\prec} W$: the workstation believes that the display is a secure channel from it;

2. U **believes** $\overset{O_c}{\prec} C$, U **believes timely**(O_c): the user believes that the smart-card's display is a secure and timely channel from the smart-card;

3. U **believes** $\overset{I_w}{\prec} U$: the user believes that the workstation's keyboard is a secure channel from him;

4. C **believes timely**(I_c): the smart-card believes that its keypad is a timely channel.

Assumptions about trust:

1. U **believes** $\forall K.(W$ **controls** $\overset{K}{\mapsto} U)$: the user believes that the workstation can choose an appropriate public key; intuitively, the user is willing to delegate to this workstation;

2. S **believes** U **controls** $\forall K.(W$ **controls** $\overset{K}{\mapsto} U)$: the certification authority trusts the user to decide whether to trust this workstation in the choice of a key; intuitively, W is believed to be a safe workstation for the user to delegate to;

3. S **believes** $\forall X.(C$ **controls** U **believes** $X)$: the certification authority trusts the smart-card to relay the user's beliefs;

4. S **believes** $\forall K.(C$ **controls** $\overset{K}{\mapsto} U)$: the certification authority trusts the smart-card to set a key for the user;

5. U **believes** W **controls** $\overset{O_w}{\prec} W$: the user trusts the workstation when it says that the display O_w is a channel from it;

6. U **believes** $\forall W.(C$ **controls** W **believes** $\overset{O_w}{\prec} W)$: the user trusts his card to pass on beliefs of the workstation;

7. W **believes** U **controls** $\overset{I_w}{\prec} U$: the workstation trusts the user when he claims to be at the keyboard.

In addition, there are a few assumptions about trust in the server; for the sake of brevity, we assume that every principal trusts the server completely.

Some of the assumptions are rather strong, and could be weakened. In particular, the assumption

$$W \textbf{ believes } U \textbf{ controls } \overset{I_w}{\prec} U$$

is a simplification of the more accurate "the workstation trusts whoever claims to be at its keyboard to identify himself properly *through the smart-card*." Our simple logic is unable to express this satisfactorily in a single formula.

3.3 The protocol analyzed

Now we discuss the protocol in detail, and show that it establishes channels (the keyboard and the display) between the user and the workstation, and that it provides a delegation key that the workstation can use on behalf of the user in dealing with other nodes:

$$U \textbf{ believes } \overset{O_w}{\prec} W, \qquad W \textbf{ believes } \overset{I_w}{\prec} U, \qquad F \textbf{ believes } \overset{K_d}{\mapsto} U$$

Step by step, we can follow the evolution of the beliefs of the participants, from the initial assumptions to these conclusions. We summarize the major deductions, and at the same time we explain the messages in the protocol.

The reasoning deals with an idealized version of the protocol, in which messages are replaced by formulas of the logic. These formulas should be believed by the principals that send them. An implementation of the protocol need not transmit these formulas literally; any unambiguous bit representation will do, and one is typically obvious from the context.

The transmission of certificates is represented explicitly. We do not show the exact routes these follow, however, as the routes do not affect the properties of the protocol—they are merely an implementation choice. In practice, one would expect certificates to be cached, so that they do not need to be transmitted or checked repeatedly.

1. $W \rightarrow C$: $\{\overset{K_d}{\mapsto} U,\ \overset{O_w}{\prec} W,\ T_w\}_{K_w^{-1}}$
   The workstation asserts that it has a public key K_d for the user and that the display O_w is a secure channel from W.

2. $S \rightarrow C$: $\{\overset{K_w}{\mapsto} W,\ T_s\}_{K_s^{-1}}$,
   $\{U \textbf{ controls } \forall K.(W \textbf{ controls } \overset{K}{\mapsto} U),\ T_s'\}_{K_s^{-1}}$
   The certification authority provides the public key of the workstation to the smart-card. At this point, the smart-card can decrypt and attribute the previous message, as well as check the freshness of the timestamp T_w. The certification authority also states that the user can let this workstation choose a delegation key. The smart-card trusts the certification authority; in the logic, we obtain:

   $$C \textbf{ believes } \overset{K_w}{\mapsto} W, \quad C \textbf{ believes } U \textbf{ controls } \forall K.(W \textbf{ controls } \overset{K}{\mapsto} U)$$

   and then also

   $$C \textbf{ believes } W \textbf{ believes } \overset{O_w}{\prec} W$$

3. $C \rightarrow U$: $W \textbf{ believes } \overset{O_w}{\prec} W \quad \mathit{on}\ O_c$
   On its display, the card provides the name of the workstation to the user; the card also gives enough information for the user to check whether the display in front of him is W's. More precisely, the smart-card states that W believes that O_w is its display (formally, that O_w is a secure channel from W). Since the user trusts the smart-card on this matter and it knows that the smart-card's display is both secure and timely, it believes the card's assertion. We can now derive:

   $$U \textbf{ believes } \overset{O_w}{\prec} W$$

because the user trusts W on $\stackrel{O_w}{\prec} W$. Thus, the workstation has authenticated itself to the user.

4. $U \rightarrow C$: $\langle \forall K.(W \textbf{ controls } \stackrel{K}{\mapsto} U), \stackrel{I_w}{\prec} U \rangle_{PIN}$ *on* I_c
   If the user wishes to proceed, he enters his PIN on the card's keypad. Thus, the user indicates that he trusts the workstation to choose a key and that he is at the keyboard I_w. He uses his personal identification number to convince the smart-card of his identity. We can prove:

   $$C \textbf{ believes } \forall K.(W \textbf{ controls } \stackrel{K}{\mapsto} U), \quad C \textbf{ believes } U \textbf{ believes } \stackrel{I_w}{\prec} U$$

   Using S's certificates and W's claims, we can also obtain:

   $$C \textbf{ believes } \stackrel{K_d}{\mapsto} U$$

5. $C \rightarrow W$: $\{U \textbf{ believes } \stackrel{I_w}{\prec} U, T_c\}_{K_c^{-1}}$
   The smart-card communicates to the workstation that the user believes he is at the keyboard. In an implementation where U and I_w are clear from context, it suffices for C to send a signed message to W, such as the delegation certificate below.

6. $S \rightarrow W$: $\{\stackrel{K_c}{\mapsto} C, \forall X.(C \textbf{ controls } U \textbf{ believes } X), T_s''\}_{K_s^{-1}}$
   The certification authority gives the smart-card's public key to the workstation, and connects this key to U. At this point, the workstation can decrypt the previous message and check its timestamp. With a few applications of the logical postulates, we prove:

   $$W \textbf{ believes } \stackrel{I_w}{\prec} U$$

   This statement means that the workstation believes that the user is at its keyboard, and hence that the user has authenticated himself to the workstation.

7. $C \rightarrow F$: $\{\stackrel{K_d}{\mapsto} U, T_c\}_{K_c^{-1}}$
   The smart-card certifies that K_d is a delegation key for the user.

8. $S \rightarrow F$: $\{\stackrel{K_c}{\mapsto} C, \forall K.(C \textbf{ controls } \stackrel{K}{\mapsto} U), T_s''\}_{K_s^{-1}}$
   The certification authority gives the smart-card's key to the file server. At this point, the file server can decrypt the certificate from the smart-card and check the timestamp in it. Furthermore, the certification

authority asserts that the card can set a key for the user, and the file server trusts the certification authority in this matter as well. Hence, we obtain:

$$F \textbf{ believes } \overset{K_d}{\mapsto} U$$

Informally, the file server has accepted the delegation key generated by the workstation, and thus the workstation can act on the user's behalf.

After this message sequence, the file server F has not heard about W. All the responsibility for checking the suitability of W rests on the user and the smart-card. This is not entirely unreasonable, as we are assuming a powerful smart-card. In the next section, we describe protocols where F participates in this checking.

4 A more realistic smart-card protocol

The smart-card design required for the protocol of the previous section is rather ambitious with today's technology. In this section, we consider a protocol that requires much less from the smart-card. Another one appears in the next section.

4.1 Concessions in smart-card design

The first feature to eliminate is the clock on the card. Having a clock on the card is particularly difficult because it requires a battery. The clock could be eliminated by having the user enter the time; it is equivalent, but much more practical, to have the workstation supply the time to the card and the user verify it on the display. This solution is not as convenient or as secure (the user will probably not check very carefully), but it works. There are two threats if the smart-card has an incorrect notion of time. First, the card could be tricked into signing a delegation certificate for some time far in the future and the workstation could then impersonate the user without the smart-card being physically present. Second, a workstation whose certificate has expired at some time in the past could convince the smart-card the certificate is still valid.

Another feature we could eliminate is the keyboard on the card. It may be quite difficult to make a small, mechanically strong keyboard that can be reached even when the card has been connected to a reader. The user could instead enter his PIN on the workstation's keyboard, which could forward the PIN in a message to the card. The danger here, though not a particularly worrisome one, is that a misbehaving workstation could capture the PIN. The workstation could give the PIN to someone who subsequently steals the card, or the workstation could use the PIN more than once on a single insertion of the card to obtain the delegation of more rôles than the user intended. These threats can be avoided entirely by having the card display a nonce secret and having the user "modify" it into the PIN with a series of '+' and 'nextdigit' operations sent via the workstation keyboard. We can view this nonce secret as an encryption key (or a one-time pad) that the card supplies to the user to establish a secure channel that the workstation cannot read or write. In this way, PIN entry goes through the keyboard but is not subject to replay.

A similar concession is to remove the display from the card, instead of the keyboard. In this case, it is straightforward for the user to enter the PIN, but it is much harder for the smart-card to identify the workstation to the user. A single LED is a partial substitute for a display.

A final compromise consists in limiting the smart-card's ability to encrypt and decrypt. We consider the extreme case where the smart-card signs one message but is not able to decrypt. In this case, it is desirable to provide a secure channel from the workstation (such as a secure card reader).

In the remainder of this section, we study a protocol where the smart-card has a display, but has neither a clock nor a keyboard, and has reduced encryption capabilities. (We leave to the reader the derivation and analysis of variants.)

These concessions in smart-card design may leave the user at the mercy of the workstation. It is therefore desirable to transfer some trust from the workstation to other principals in the distributed environment.

The simplest choice is to give a more prominent rôle to the principals that check delegation certificates, such as the file server F. Thus, F takes into account W's identity before accepting a delegation key for U. Moreover, F may grant some requests from W on behalf of U, and not others (as in the general approach to delegation of [8]). The user obtains no real guarantee of the identity of the workstation, since the card cannot decrypt. However, the user can be sure that his authority is not delegated inappropriately, because of the check performed by F. This is the approach we adopt in the following protocol.

In the next section, we discuss a more elaborate design, where dedicated *trusted agents* assist the smart-card in the process of delegation.

4.2 Notation and assumptions

Some additional notation is needed:

- K_{cu} is a short secret nonce generated by the card, to protect the entry of the user's PIN—as the notation suggests, we view the secret as an encryption key for the PIN;
- L_r is the smart-card reader; and
- $\{T_b\}_{K_b^{-1}}$ is a timestamp signed by B, a trusted time service; B is trusted never to sign a timestamp for a time in the future; W obtains the certified timestamp by whatever means, and may check T_b.

The following are the main novelties in the initial assumptions; we omit a full list. In these, it is convenient to use the name W' for the workstation that controls the channel L_r (with any luck, of course, W' is the intended W). We need this notation because the user must reason about the workstation behind L_r before believing it is W.

1. C **believes** $\overset{L_r}{\prec} W'$, C **believes timely**(L_r): the smart-card believes that the smart-card reader provides a timely, secure channel to the workstation W';
2. C **believes** $C \overset{K_{cu}}{\leftrightarrow} U$, C **believes fresh**$(C \overset{K_{cu}}{\leftrightarrow} U)$: the smart-card has a secret, to be used to hide the PIN from the workstation; this secret is a nonce, and hence its mention in a message proves the freshness of this message.
3. U **believes** W' **controls** $\overset{O_w}{\prec} W$: the user trusts the workstation behind the smart-card reader to give its name correctly, asserting that the display is a secure channel from W—the user may reject outrageous names, though; this assumption is strong, since the user gets no real guarantee of the workstation's identity;
4. F **believes fresh**$(\{T_b\}_{K_b^{-1}})$: the file server believes in the timeliness of the certified timestamp; B's signature convinces F of the validity of T_b, even though C does not have a clock, because B is trusted not to sign future timestamps.

4.3 The protocol analyzed

The protocol presented here achieves the same properties as the one for ultimate smart-cards, namely,

$$U \textbf{ believes } \overset{O_w}{\prec} W, \qquad W \textbf{ believes } \overset{I_w}{\prec} U, \qquad F \textbf{ believes } \overset{K_d}{\mapsto} U$$

This amounts to mutual authentication and delegation. However, stronger assumptions are required here, particularly for the first conclusion.

The messages and the deductions can be explained thus:

1. $W \rightarrow C$: $\overset{O_w}{\prec} W$ *on* L_r
   The workstation names itself and its display.

2. $C \rightarrow U$: W' **believes** $\overset{O_w}{\prec} W, C \overset{K_{cu}}{\leftrightarrow} U$ *on* O_c
   The card passes the workstation's message on to the user. It also provides a one-time pad K_{cu}. Notice that the user has no assurance that the named workstation is actually the workstation in front of him. However, he can be sure that he is delegating authority only to the machine named. The user simply trusts the workstation to give its name correctly. Thus, we have:

$$U \textbf{ believes } \overset{O_w}{\prec} W$$

3. $U \rightarrow C$: $\{\overset{I_w}{\prec} U, \forall K.(W \textbf{ controls } \overset{K}{\mapsto} U), C \overset{K_{cu}}{\leftrightarrow} U\}_{Kcu}$
   The user enters his PIN. The PIN is not entered directly; instead, the user types a sequence of keys which modify the displayed value K_{cu} until the PIN is displayed. Thus, the user asserts that he is at the keyboard, and gives jurisdiction to the workstation over the choice of a key. The key K_{cu} appears inside the message as proof of timeliness. (In an implementation, the use of K_{cu} as a one-time pad suffices as proof of timeliness.)

4. $C \rightarrow W$: $\{U \textbf{ believes } \overset{I_w}{\prec} U, \{T_b\}_{K_b^{-1}}\}_{K_c^{-1}}$
   As in the previous protocol, the smart-card communicates to the workstation that the user believes he is at the keyboard. The certified timestamp, which may have been obtained via W, is included as proof of timeliness.

5. $S \rightarrow W$: $\{\overset{K_c}{\mapsto}C, \forall X.(C \textbf{ controls } U \textbf{ believes } X), T_s''\}_{K_s^{-1}}$
The certification authority provides the smart-card's public key to the workstation and certifies that the smart-card is allowed to transmit the user's beliefs. At this point the workstation can interpret the previous message. As in the previous protocol, the user has authenticated to the workstation:
$$W \textbf{ believes } \overset{I_w}{\prec}U$$

6. $C \rightarrow F$: $\{U \textbf{ believes } W \textbf{ controls } \overset{K_d}{\mapsto}U, \{T_b\}_{K_b^{-1}}\}_{K_c^{-1}}$
In this delegation certificate, the smart-card asserts that the user has delegated to the workstation: it says that the user believes that the workstation has jurisdiction over setting K_d as his delegation key. The certificate includes $\{T_b\}_{K_b^{-1}}$ as proof of timeliness.

7. $W \rightarrow F$: $\{\overset{K_d}{\mapsto}U, \{T_b\}_{K_b^{-1}}\}_{K_w^{-1}}$
The workstation asserts that K_d is a delegation key for U.

8. $S \rightarrow F$: $\{\overset{K_w}{\mapsto}W, T_s\}_{K_s^{-1}}$,
$\{\forall K.(U \textbf{ controls } W \textbf{ controls } \overset{K}{\mapsto}U), T_s'\}_{K_s^{-1}}$,
$\{\overset{K_c}{\mapsto}C, \forall X.(C \textbf{ controls } U \textbf{ believes } X), T_s''\}_{K_s^{-1}}$
The certification authority provides the public keys of the smart-card and the workstation. It certifies that the user can delegate to this workstation, and that the smart-card is allowed to transmit the user's beliefs. Using the previous messages, we obtain the desired delegation result:
$$F \textbf{ believes } \overset{K_d}{\mapsto}U$$

It is not strictly necessary to include the key K_d in the delegation certificate, as we have. However, the mention of the key makes it simple for the workstation to renounce delegated powers when they are no longer needed, by forgetting the matching secret key K_d^{-1}. Thus, the user is protected against future compromise of the workstation, even if the delegation certificate has a long lifetime.

As usual, the number of messages can be reduced by caching commonly used certificates. Furthermore, the smart-card need perform only one signing operation, and this can be done while the user enters his PIN. Hence the card need not be particularly fast. The total complexity of the protocol has increased slightly, but the demands on the smart-card have decreased. This protocol, or similar ones, may well be practical.

5 A protocol with trusted agents

An alternative approach to reducing the demands on the smart-card is based on the use of trusted agents. We discuss this solution here.

5.1 Trusted agents

An on-line trusted agent can relieve the smart-card from the elaborate rituals of generating timestamps and verifying certificates. As this trusted agent can check that W is a suitable workstation for U, this burden is removed from principals such as F. Moreover, a trusted agent simplifies the process of revocation for a compromised workstation—a trusted agent may be a convenient place for a workstation black list. The workstation and the trusted agents can check on one another.

As we envision them, these trusted agents are dedicated, physically protected machines. There would be a large number of trusted agents widely dispersed. Each trusted agent assists a community of users under a single domain or management. Any such arrangement reduces availability (when all replicas are down or inaccessible, the user cannot work) and lessens security (the agents are an attractive target).

If the smart-card can execute only the DES algorithm [4], then the trusted agent will get access to the user's private key during the login process. It can still be arranged that compromise of a trusted agent will not permit the impersonation of all users who trust it—only of those who use it while it is compromised.

If the smart-card can perform the RSA signing operation, a protocol can be obtained whereby the trusted agent cannot impersonate the user. The compromise of a trusted agent does not destroy security per se. In the solution explored in the rest of this section, the smart-card will sign anything the workstation gives it, but no one will believe anything signed by the smart-card without a certificate from a suitable trusted agent. As in the previous protocol, the user obtains no real guarantee of the identity of the workstation, since the card cannot decrypt. However, the user can be sure that his authority is not delegated inappropriately, because of the check performed by the trusted agent.

5.2 Notation and assumptions

Let A be a trusted agent, K_a his public key, and T_a a timestamp he generates.

The most important new assumptions are:

1. S **believes** $\forall K \forall W.(A$ **controls** W **believes** $\stackrel{K}{\mapsto} U)$,
   S **believes** $\forall K \forall W.(A$ **controls** U **controls** W **controls** $\stackrel{K}{\mapsto} U)$: the certification authority believes that the user is in the domain of the trusted agent A; the formulas represent consequences of this belief.

2. A **believes** $\forall K.(U$ **controls** W **controls** $\stackrel{K}{\mapsto} U)$: the trusted agent A believes that U can delegate to W.

We treat A as different from B, the trusted time provider, although A and B could obviously be implemented by a single node.

5.3 The protocol analyzed

Many of the messages are identical to those of the previous protocol. We discuss only the changes.

1. $W \rightarrow A$: $\{\stackrel{K_d}{\mapsto} U, T_w\}_{K_w^{-1}}$
   The workstation asserts that K_d is a delegation key for U.

2. $S \rightarrow A$: $\{\stackrel{K_w}{\mapsto} W, T_s\}_{K_s^{-1}}$
   The trusted agent consults a certificate that contains W's key.

3. $W \rightarrow C$: $\stackrel{O_w}{\prec} W \quad on\ L_r$

4. $C \rightarrow U$: W' **believes** $\stackrel{O_w}{\prec} W, C \stackrel{K_{cu}}{\leftrightarrow} U \quad on\ O_c$

5. $U \rightarrow C$: $\{\stackrel{I_w}{\prec} U, \forall K.(W$ **controls** $\stackrel{K}{\mapsto} U), C \stackrel{K_{cu}}{\leftrightarrow} U\}_{K_{cu}}$

6. $C \rightarrow W$: $\{U$ **believes** $\stackrel{I_w}{\prec} U, \{T_b\}_{K_b^{-1}}\}_{K_c^{-1}}$

7. $A \rightarrow F$: $\{W$ **believes** $\stackrel{K_d}{\mapsto} U, U$ **controls** W **controls** $\stackrel{K_d}{\mapsto} U, T_a\}_{K_a^{-1}}$
   The trusted agent checks that the workstation is a reasonable machine for the user to trust. It signs a certificate to that effect, including the delegation key for the subsequent session. The trusted agent states that W has chosen K_d, and that U can let W use K_d as a delegation key.

8. $C \rightarrow F$: $\{U$ **believes** W **controls** $\overset{K_d}{\mapsto} U$, $\{T_b\}_{K_b^{-1}}\}_{K_c^{-1}}$
The file server checks that values of U and W match those in the previous message. This check ensures that both the user and the trusted agent refer to the same workstation.

9. $S \rightarrow F$: $\{\overset{K_a}{\mapsto} A, T'_s\}_{K_s^{-1}}$
$\{\forall K \forall W.(A$ **controls** W **believes** $\overset{K}{\mapsto} U)$,
$\forall K \forall W.(A$ **controls** U **controls** W **controls** $\overset{K}{\mapsto} U)$, $T''_s\}_{K_s^{-1}}$
$\{\overset{K_c}{\mapsto} C, \forall X.(C$ **controls** U **believes** $X)$, $T'''_s\}_{K_s^{-1}}$
The file server obtains certificates for A and C.

The result is the usual one: mutual authentication and delegation.

6 Conclusions

Authentication protocols that use smart-cards are a significant improvement over those that use simple passwords. We have described a few smart-card protocols and the guarantees they offer. We feel that the use of a formalism has helped us elucidate and compare some of the subtle trust relations that underly these protocols.

A trade-off is inevitable, between the trust that the user needs to place in the environment, and the power, cost, and size of his smart-card. Each of the protocols—and there are others—has its own problems and addresses specific threats, with specific technological requirements.

Appendix: The logic

In the analysis of smart-card protocols, we apply a logic of authentication. The notation is as given in Section 2; here we give a few of the main rules of inference and briefly explain how to use them. The logic is presented in [1]; a Kripke semantics is currently being developed by M. Tuttle and M. Abadi.

Rules of inference

We manipulate formulas of the logic with rules of inference, such as the following.

- The jurisdiction rule reflects that if P believes that Q is an authority on X then P trusts Q on the truth of X:

$$\frac{P \textbf{ believes } Q \textbf{ controls } X, \quad P \textbf{ believes } Q \textbf{ believes } X}{P \textbf{ believes } X}$$

- The public-key message-meaning rule concerns the interpretation of encrypted messages:

$$\frac{P \textbf{ believes } \overset{K}{\mapsto} Q, \quad P \textbf{ sees } \{X\}_{K^{-1}}}{P \textbf{ believes } Q \textbf{ said } X}$$

  That is, if P believes that the key K is Q's public key and P sees X encrypted under K's inverse, then P believes that Q once said X.

- A similar message-meaning rule applies to links:

$$\frac{P \textbf{ believes } \overset{L}{\prec} Q, \quad P \textbf{ sees}_L\ X}{P \textbf{ believes } Q \textbf{ said}_L\ X}$$

- A nonce-verification rule expresses the check that a part of a message is recent, and hence that the sender still believes in the message:

$$\frac{P \textbf{ believes } \textbf{fresh}(X), \quad P \textbf{ believes } Q \textbf{ said } X}{P \textbf{ believes } Q \textbf{ believes } X}$$

  That is, if P believes that X could have been uttered only recently and that Q once said X, then P believes that Q has said X recently, and hence that Q believes X. A variant of this rule is sometimes useful:

$$\frac{P \textbf{ believes } \textbf{fresh}(Y), \quad P \textbf{ believes } Q \textbf{ said } (X, Y)}{P \textbf{ believes } Q \textbf{ believes } X}$$

  For the sake of simplicity, we use this rule only when X is cleartext, that is, it has no subformulas of the form $\{Y\}_K$. A similar remark applies to all other rules that introduce the **believes** operator.

- Another way to guarantee timeliness is by using timely communication links:

$$\frac{P \textbf{ believes } \textbf{timely}(L), \quad P \textbf{ believes } Q \textbf{ said}_L\ X}{P \textbf{ believes } Q \textbf{ believes } X}$$

On quantifiers in delegations

Delegation statements usually mention one or more variables. For example, the user U may let the workstation W generate an arbitrary delegation key. We can express this as

$$U \textbf{ believes } W \textbf{ controls } \stackrel{K}{\mapsto} U$$

Here the key K is universally quantified, and we can make explicit this quantification by writing

$$U \textbf{ believes } \forall K.(W \textbf{ controls } \stackrel{K}{\mapsto} U)$$

For complex delegation statements, it is generally necessary to write quantifiers explicitly in order to avoid ambiguities. In some previous works on the logic, this need was not recognized, as in fact it did not arise. (There were no nested jurisdiction statements.) This need does arise in the proofs above.

Our formal manipulation of quantifiers is quite straightforward. All we use is the ability to instantiate variables in jurisdiction statements, as reflected by the rule

$$\frac{P \textbf{ believes } \forall V_1 \ldots V_n.(Q \textbf{ controls } X)}{P \textbf{ believes } Q' \textbf{ controls } X'}$$

where Q' **controls** X' is the result of simultaneously instantiating all of the variables $V_1, \ldots, V_n$ in Q **controls** X.

Protocol analysis

Authentication protocols are typically described by listing their messages in the form

$$P \rightarrow Q : \mathit{message}$$

This denotes that P sends the message to Q. Occasionally, it is stated that the message follows a particular route, such as a secure channel.

The message is presented in an informal notation designed to suggest the bit-string that a particular concrete implementation would use. In the interest of formal analysis, we rewrite each message as a logical formula. For instance, the protocol step

$$A \rightarrow F : K_a \textit{ on link } L_1$$

may tell F, who knows that L_1 is a secure channel from A, that K_a is A's public key. This step should then be idealized as

$$A \rightarrow F : \stackrel{K_a}{\mapsto} A \quad on\ L_1$$

We annotate idealized protocols with logical formulas, much as in a proof in Hoare logic ([H]). We write formulas before the first message and after each message. The main rules for deriving legal annotations are:

- if X holds before the message $P \rightarrow Q : Y$ then both X and Q **sees** Y hold afterwards;
- if X holds before the message $P \rightarrow Q : Y \quad on\ L$ then both X and Q $\textsf{sees}_L$ Y hold afterwards;
- if Y can be derived from X by the logical postulates then Y holds whenever X holds.

An annotation of a protocol is like a sequence of comments about the beliefs of principals and what they see in the course of authentication, from initial assumptions to conclusions.

Acknowledgements

Jim Saxe invented the method for entering a PIN by modifying a nonce with a series of '+' and 'nextdigit' operations. Dorothy Denning and Roger Needham read drafts of this paper. Cynthia Hibbard provided comments on the presentation.

References

[1] M. Burrows, M. Abadi, and R.M. Needham. A Logic of Authentication, *Proceedings of the Royal Society of London A* Vol. 426, 1989, pp. 233–271. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, February 1989.

[2] CCITT. CCITT Blue Book, Recommendation X.509 and ISO 9594-8: The Directory-Authentication Framework. Geneva, March 1988.

[3] D. Chaum and I. Schaumüller-Bichl, editors. Smart Card 2000: The Future of IC Cards. Proceedings of the IFIP WG 11.6 International Conference on Smart Card 2000: The Future of IC Cards, Laxenburg, Austria, October, 1987. North-Holland, Amsterdam, 1989.

[4] National Bureau of Standards. Data Encryption Standard. Fed. Inform. Processing Standards Pub. 46. Washington DC, January 1977.

[5] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory* IT-22, No. 6, November, 1976, pp. 644–654.

[6] M. Gasser, A. Goldstein, C. Kaufman, B. Lampson. The Digital Distributed System Security Architecture. Proceedings of the 1989 National Computer Security Conference, Baltimore, 1989, pp. 305-319.

[7] U. Feige, A. Fiat, A. Shamir. Zero Knowledge Proofs of Identity. Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, New York, 1987, pp. 210–217.

[8] M. Gasser, E. McDermott. An Architecture for Practical Delegation in a Distributed System. Proceedings of the 1990 IEEE Symposium on Security and Privacy, Oakland, 1990, pp. 20–30.

[9] C.A.R. Hoare. An Axiomatic Basis for Computer Programming, *CACM* Vol. 12, No. 10, October 1969, pp. 576–580.

[10] S.P. Miller, C. Neuman, J.I. Schiller, and J.H. Saltzer. Kerberos Authentication and Authorization System. *Project Athena Technical Plan* Section E.2.1, MIT, July 1987.

[11] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *CACM* Vol. 21, No. 12, December 1978, pp. 993–999.

[12] R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems, *Communications of the ACM* Vol. 21, No. 2, February 1978, pp. 120-126.