Robust and Secure Password and Key Change Method*

Ralf Hauser†, Philippe Janson†, Refik Molva‡, Gene Tsudik†, Els Van Herreweghen†

November 4, 1993

*DRAFT–do not distribute.

†IBM Research Laboratory, CH-8803 Rüschlikon, Switzerland. {*rah,pj,gts,evh*}*@zurich.ibm.com*

‡EURECOM Institute, Sophia Antipolis, 06560 Valbonne, France. *molva@eurecom.fr*

Robust and Secure Password and Key Change Method

Abstract

This paper discusses issues and idiosyncrasies associated with changing passwords and keys in distributed computer systems. Current approaches are often complicated and fail to provide the desired level of security and fault tolerance. A novel and very simple approach to changing passwords/keys is presented and analyzed. It provides a means for human users and service programs to change passwords and keys in a robust and secure fashion.

1 Introduction: Changing One's Password

Much effort has recently gone into securing user access to computer systems over insecure communication lines from untrusted (or partially-trusted) workstations and other end-user devices. In a distributed and dynamic network environment, solutions are often based on the use of a highly-secure and trusted entity called an Authentication Server (AS). An AS processes authentication requests by acting as a trustworthy intermediary. As such, the AS has access to and control over the authenticating secrets of all its principals, be they human users or service programs. (Hereafter, we use the term *principal* to refer to both human users and service programs.)

Typically, a principal's password is a fairly long-term secret, i.e., most principals are not required to change their passwords more often than, say, once a month. When a principal wishes to change its password, an appropriately authenticated exchange with the AS must take place. Obviously, any such exchange must meet some basic security requirements such as resistance to guessing attacks and replays of change-password requests. Another desirable feature is the *correctness* of the change-password protocol. We say that a change-password protocol is correct is it provides a guarantee of state synchronization between the two parties involved (AS and principal) in the presense of possible crashes and network failures.

This last issue is not directly relevant to the protocol security and is thus frequently overlooked or not given enough consideration. However, it has to do with the robustness and usability of the protocol which is, in the end, an issue of great importance to the end-users.

Supposing that a principal would like to change its password from K_{old} to K_{new}, six outcomes of the password change protocol are possible:

	Principal believes its secret is	AS believes principal's secret is
1	K_{new}	K_{new}
1	K_{new}	K_{new}
2	K_{old}	K_{old}
3	K_{old}	K_{new}
4	K_{new}	K_{old}
5	K_{old}	K_{unk}
6	K_{new}	K_{unk}

The first two outcomes are considered normal and desirable. In the first case, a successful password change takes place, i.e., the AS records the new password and the principal receives the change confirmation. In the second case, the new password is rejected for some reason and the principal is informed.

The rest of the cases represent anomalous situations and must be avoided at all costs. Case 3 may occur when a change acknowledgement is lost or when the AS crashes after making a change but before sending out the acknowledgement. The next situation (case 4) may take place if the acknowledgement is somehow spoofed, i.e., an adversary is able to compose a fake confirmation message.[1] Case 5 is the result of an adversary successfully manipulating the protocol is such a way that the AS changes the principal's key to some value (K_{unk}) unknown to or, at least, not intended by, the principal. In fact, case 5 can occur without any activity on the part of the principal; the adversary may simply concoct the entire protocol. Finally, case 6 is very similar to case 5 except that here the principal is actually trying to run the protocol and the adversary not only succeeds in *convincing* the AS to change the principal's password to some K_{unk}, but also manages to convince the principal that the AS accepted K_{new}.

In the rest of this paper we review the current state-of-the-art (exemplified by Kerberos) and go on to develop a protocol that is at the same time simple, robust and secure. The protocol is explicitly constructed to handle anomalous behavior (i.e., events that can lead to cases 3-6 above) of the network and arbitrary failures of the components involved.

2 The *Kerberos* Approach

One of the most popular network security solutions in use today is the Kerberos Authentication and Key Distribution Server originated at MIT[2] and later integrated in the OSF DCE product[3]. Among its multitude of features, Kerberos includes a change password (CPW) protocol implemented by the kpasswd command which is part of the standard Kerberos distribution package. The actual protocol varies slightly between successive versions of Kerberos. The CPW protocol for Kerberos versions 4 and 5, is illustrated in Figure 1.

The general course of events in Kerberos CPW protocols is as follows (unless noted otherwise, the following description applies to both versions):

1. Initially, the requesting principal engages in an authentication exchange with the AS to obtain credentials for the Admin server. An Admin server, in Kerberos parlance, is a logically distinct entity which is responsible for the maintenance of all information about principals. (In practice, Admin server is almost always co-located with an AS).

[1] We do not consider the case when an AS fails to make the change but acknowledges it nonetheless.

This initial exchange with the AS requires the principal to provide its current password. The credentials obtained consist of a temporary key K_{cpw} to be shared with the Admin server and a special CHANGEPW ticket (T_{cpw}) with a reasonably short lifetime, typically set to one minute.

2. The second part of the protocol consists of an exchange between the principal and the Admin server. In the course of this exchange, the principal authenticates itself to Admin and sends along the new password, encrypted with the shared key K_{cpw}. In the version 5 protocol, a complete mutual authentication between principal and Admin takes place before the principal sends its encrypted password; while in the version 4 protocol, the same message carries the principal's authentication and the (encrypted new password.

3 Discussion

The Kerberos CPW protocol is, essentially, capability-based. The capability is embodied in the CHANGEPW ticket which gives the principal the right to change its password. It *does not* restrict the number of times a principal may do so using the same capability. Thus, in general, a principal can use the same ticket multiple times (of course, as long as the ticket does not expire.) This feature is not a drawback in and of itself but it separates two important events that should ideally go hand-in-hand:

- The verification of the previous state of the principal (i.e., making sure that the principal knows the old key/password)

  and

- The verification of the new state of the principal (i.e., making sure that the new password is actually supplied by the true principal and is acceptable to the Admin server) plus the actual database change.

As is typical with capability-based approaches, revocation is very difficult if not impossible and is the source of many problems. Of course, a capability tightly bound by time (i.e., one minute in Kerberos) is not as dangerous, but on the other hand, the usual approach of using black-lists to provide for revokation is also not very effective because the assembly and distribution of such a list already might take a significant fraction of this short time.

The existence of the capability-like CHANGEPW ticket itself is dangerous because Kerberos stores its tickets in special cache files on disk. Within the lifetime of a CHANGEPW ticket (which is by default a minute) a trojan horse program could create an additional message changing the password to a value unknown to the principal but known to an adversary controlling the trojan horse. A similar problem could occur if the principal left the terminal unattended just after changing the password. Then, an adversary could walk by and, on the victim's behalf, run kpasswd. Since there still can be a valid CHANGEPW ticket cached locally, kpasswd could bypass prompting the principal (actually, the adversary) for the old password and use the CHANGEPW ticket to change the principal's password to anything the adversary chooses. It should be noted, that, in practice, the Kerberos kpasswd command promptly destroys the CHANGEPW ticket upon the completion of the protocol so that the described attack is mainly theoretical.

More often than not, a principal wanting to change its password already obtained a Ticket-Granting-Ticket (TGT) from the AS by going through the initial login procedure. Therefore, a principal already has a strong key that can be used to protect all subsequent communication from, among other things, password-guessing attacks. The Kerberos approach doesn't take advantage of this. Consequently, the initial message (request for CHANGEPW ticket) in the Kerberos protocol is susceptible to password-guessing attacks in case when pre-authentication is used. Also in the case when the pre-authentication feature is not used, the reply from the AS is still susceptible to password-guessing attacks (since the password-derived key K_{old} is used to distribute K_{cpw} in CHANGEPW ticket.)

Another concern with the Kerberos approach is the number of protocol messages involved. As mentioned earlier, fault tolerance is of utmost importance when it comes to changing one's password. Therefore it is beneficial to minimize the number of protocol exchanges. In Kerberos, four (in version 4) or six (in version 5) messages are neeeded between the AS/Admin and the principal's end-system. This is a direct consequence of the Kerberos AS and the Admin server being logically distinct entities: the principal has to first authenticate itself to the AS before requesting the actual password change from Admin. As will be described below, it is possible and desirable to reduce the number of messages to just two, by authenticating the CPW request directly with the principal's password (or key) instead of a short-term key.

Perhaps one of the main issues is that in case of a loss of one of the (4 or 6) protocol messages, no algorithm is given to allow for automatic recovery without resorting to off-line means. In particular, if the last (number 4 or 6 depending on the version) protocol message is lost or if the Admin server crashes right before sending out the last message, the protocol terminates in a state without automatic recovery. In other words, one has to resort to extra-protocol means such as trying to log in anew with, say, the old key and, if that fails, with the new key. Alternatively, one can try to change password once again and hope that the change has not gone through.

Despite the above criticism, Kerberos provides a reasonably workable and secure solution. However, there are several undesirable characteristics where improvement is possible:

- Too many protocol messages
- Vulnerability to password-guessing attacks
- No provisions for *graceful* recovery in case of message loss and/or component failures.

The protocol presented in the next section addresses these issues.

4 Solution

NOTE: For the sake of uniformity, the term *key* is used in reference to both passwords and keys.

Our approach addresses the following requirements:

- The change request should contain authentication of the sender. In case when the request originates with a human user at a remote workstation, the user must provide old (current) password in order to prevent fraudulent password changes when a workstation is left unattended.
- The request itself must be authenticated, i.e. the AS must be able to establish the integrity of the new key (K_{new}) defined in the request.
- The AS has to confirm the outcome of the password change to the requesting principal. The acknowledgement itself must be authenticated.

- The AS must be able to identify retransmissions of previously processed requests and to issue acknowledgments for such retransmissions. This is necessary whenever the original acknowledgement is lost and the principal resubmits the change request. (There is no danger in that as long as the acknowledgements remain identical.)
- An adversary should not be able to gather any useful information from replaying a *stale* request message. This should hold even in the event that the principal makes a serious error of reusing passwords.

4.1 Assumptions

The following assumptions are made hereafter:

1. The principal does not start "believing" in K_{new} until successful completion of the CPW protocol. Of course, K_{new} is also not used if a negative acknowledgement rejecting K_{new} is received (e.g., because it is trivial, predictable, or otherwise unacceptable to the AS.)
2. If the CPW protocol does not terminate normally, the principal is capable of remembering the new and old password until the next attempt, i.e., the resumption of CPW protocol. In other words, is something abnormal takes place and the protocol does not complete, the principal does not **abandon** the change; instead the procedure is re-tried at some later time.
3. The AS needs to be no more than *single-state*. In other words, it only has to remember one (current) password per principal and does not have to keep any password history. Furthermore, it has a fairly accurate clock. Fairly accurate means that it is accurate with respect to the frequency of CPWs which happen infrequently, i.e., daily, weekly or monthly, but not every minute or hour.
4. The hosts or workstations (where CPW requests originate) also possess fairly accurate clocks. If not, the principal's wristwatch or wall-clock readings are good enough.
5. It is not taken for granted that the principal shares a strong key with the AS. (Such a key could be obtained during initial login by the principal, and cached locally). However, if such a strong key (K_{sso}) is present, it may be used to increase the security of the change-password protocol.

4.2 Basics

The basic idea of the protocol is to construct an atomic "flip-flop" request in such a way that a change-key request from principal P to change its key (password) from K_{old} to K_{new} can be verified and honored by the AS **independent** of whether the AS knows only K_{old} or only K_{new}. In other words, the AS is able to recognize, authenticate and acknowledge a retransmission of the CPW request, even after having discarded K_{old} and replaced it with K_{new}. This feature enables the AS and P to resynchronize even in case a positive acknowledgment from the AS is lost or the AS has crashed at an inopportune moment.

If the initial request goes unacknowledged, the principal simply retransmits the request. In this case, the AS knows either K_{old} or K_{new} depending on whether it was the request or the acknowledgement that was lost. In any case, the flip-flop construction of the request enables the AS to process the request message correctly regardless of the current state.

As soon as the AS receives a well-formed authentic request, it replies with an acknowledgement. If the AS receives the same request again, the acknowledgement must have been lost, thus the database is left untouched and another acknowledgement (the exact copy of the original one) is re-generated.

The result of this simple protocol is that there may be a temporary uncertainty on the side of the principal as to the current state of the AS, but this requires no action by the principal beyond retransmitting the original request. The principal may do so ad nauseum but, eventually, when communication is re-established, the first acknowledgement re-synchronizes the two sides.

4.3 Protocol Description

The actual protocol is depicted in Figure 2. It consists of only two messages. In the first message (REQ_CPW), P transmits two tickets to the AS: a ticket T$(K_{new})K_{old}$ containing the new key K_{new}, and sealed with K_{old}; and a "sanity check" ticket T$(K_{old})K_{new}$ sealed with K_{new}, and containing K_{old}. The ticket expressions are similar to those used in the KryptoKnight Authentication and Key Distribution Service [8]. (See also [6, 7, 5].)

If the request is well-formed and authentic, AS replies with an authenticated acknowledgment REP_CPW which can take on two flavors: ACK (*accept*) or NACK (*reject*). The AS generates a NACK if only if K_{new} is not acceptable for some reason (e.g., predictable password). However, K_{new} must still satisfy the "flip-flop" property of the REQ_CPW. In other words, AS replies (be it with an ACK or a NACK) only if REQ_CPW is genuine.

4.3.1 Details of the REQ_CPW message

As shown in the ticket expression of T$(K_{old})K_{new}$ in Figure 2, the function "g" provides for asymmetry between the two tickets in such a way that an adversary cannot swap the two tickets and convince a server to switch back to the old key. We note that "g" must be asymmetric, otherwise manipulation of the plaintext N2 (e.g. reciprocal value or XOR with N1) would re-enable the above swapping attack.

With the above requirement in mind, one possibility is g=(x+1). Another one is g=$\{$N2$\}K_{new}$.

If the principal already obtained a strong key K_{sso} (perhaps during the initial login) then "g" could depend on K_{sso}. For example we can set g=$\{N2\}K_{sso}$. This increases the resistance of the protocol against password guessing attacks (since an adversary would have to break K_{sso} before attacking K_{old} and/or K_{new}).

The first nonce N1 is chosen at random. In contrast, N2 is set to the current time. This does not require synchronized clocks because the maximum workstation clock-skew is assumed to be smaller than the frequency of key changes. An adversary could still set a workstation's clock to some random time in the future. The AS would reject the REQ_CPW because of the wrong timestamp, but the adversary could replay it later when the timestamp becomes *ripe*. However, according to the second assumption (in Section 4.1), the principal does not abandon the change but retries at a later time. Eventually, a re-try will be acknowledged by the AS.

All in all, two items provide for AS-principal synchronization: K_{old} and N2 which represents a timestamp.

4.3.2 Processing REQ_CPW

1. Having received a REQ_CPW, the AS first extracts K'_{new} from T$(K_{new})K_{old}$, using K_{as} – the princi-

pal's key currently in the database. (Assuming, so far, that $K_{as} = K_{old}$.)

2. Then, using K'_{new}, the AS extracts K'_{old} from $\mathrm{T}(K_{old})K_{new}$.

3. If $K'_{old} = K_{as}$, the AS is assured that it still has the principal's old key and K'_{new} is the new key intended. AS then stores K_{new} in the database and sends back a positive acknowledgement, i.e., REP_CPW(accept) the format of which is described below.

4. If $K'_{old} <> K_{as}$, the message could still be a re-try from a principal who didn't receive the original REP_CPW(accept) message for a successful key-change. In this case, K_{as} would already be the same as K_{new}.

   (a) Using K_{as}, the AS extracts K'_{old} from $\mathrm{T}(K_{old})K_{new}$.

   (b) Using K'_{old}, the AS extracts K'_{new} from $\mathrm{T}(K_{new})K_{old}$.

   (c) If $K'_{new} = K_{as}$, the AS is assured that it already has the new key stored in the database; it then generates REP_CPW(accept) for requesting principal.

4.3.3 Acknowledgements

Acknowledgments for the following cases must be provided:

- key successfully changed to K_{new} as a result of either this or some previous REQ_CPW. (In the latter case, the present REQ_CPW is a re-transmission.)

- K_{new} is unacceptable but REQ_CPW is well-formed, i.e., its token structure is correct.

The acknowledgement message has the following form:

$$\mathrm{REP\_CPW} = \mathrm{token}(K_{old}) \text{ containing [accept/reject, REQ\_CPW]}$$

This token is an integrity check of the above two components of REP_CPW.

An incorrect or malformed REQ_CPW is one where:

- K_{old}/K_{new} do not satisfy the "flip-flop" structure described above,

  or

- the timestamp represented by N1 is unacceptable, i.e., outside the limits of maximum acceptable clock skew.

Malformed (not authentic) REQ_CPWs are not acknowledged at all. A cleartext error message is a possible alternative. However, any kind of authenticated acknowledgment in response to an incorrect REQ_CPW is out of the question. This is because doing so would require using the principal's current stored key which would present an opportunity for a known plaintext attack (the AS would become an *oracle*, see [4]). Therefore, the mechanism on the principal's side must at least provide for an error message which after a certain number of unanswered REQ_CPW (timeout) checks for the general availability of the AS and suggests resorting to off-line means for re-synchronization.

Obviously, the acknowledgment (REP_CPW) must be protected. If it is not protected by a strong integrity check, an adversary could trap the original REQ_CPW, prevent it from reaching the AS and convince the principal that the change has taken place. The key used to protect REP_CPW can be any of: K_{new}, K_{old} or K_{sso}. However, if K_{sso} is used, the previous flow, REQ_CPW, must additionally contain the ticket with K_{sso}. One problem with using K_{new} is when the AS rejects K_{new} for some reason (e.g., weak key) it cannot very well use the same key it just rejected to compute the integrity check. Therefore, K_{old} must be used in this case. Alternatively, for the sake of uniformity and simplicity, K_{old} can be used in all cases (i.e., ACK or NACK).

5 Some Remarks on the Security of the Proposed Protocol

In addition to the threats already addressed in the protocol description above, the following venues of attacking the protocol must be considered:

- If the adversary eavesdrops on a REQ_CPW, the "flip-flop" feature allows for an off-line key-search attack. This attack is possible because the very same structure of REQ_CPW that allows the AS to verify K_{new} and K_{old} allows the adversary to verify its guesses by iterating through the key space.

  If the function "g" involves K_{sso}, the adversary must additionally eavesdrop on the initial login flows containing a ticket sealed with K_{old}. Therefore, this attack is not applicable since the adversary could verify its password guesses directly on any other subsequent message security of which depends on the initial SSO without needing REQ_CPW at all. If "g" does not depend on K_{sso}, the threat remains valid. However, the initial login (SSO) protocol is still the weakest point for the following reasons:

  - *unassisted login* without smartcards or similar devices is at least as vulnerable as the present protocol.
  - principals log in much more often than they change passwords thus affording much greater opportunity for adversaries.

6 Protocol Correctness

In this section we analyze the correctness of the proposed protocol.

6.1 Assumptions

To aid in our analysis, the following assumptions are made:

- **Assumption 1:** The principal generates a different key at every execution of the change password protocol. Therefore the password history has no cycles. An execution of CPW means a protocol run until successful completion or synchronization by out-of-band means.

- **Assumption 2:**
  Each message will be received after a finite number of retransmission attempts.

- **Assumption 3:**
  To simplify the proof of correctness, we will assume that the AS only sends positive acknowledgments (*reject*ed requests are not acknowledged).

6.2 Idealized Protocol

Before we proceed to the formal analysis we need an idealized and formal representation of the protocol.

Idealized Representation of the protocol:
The protocol consists of two communicating state machines U and AS depicted in figure 3, that respectively represent the behavior of the change password program on the principal's side and the behaviour of the AS in response to the presented change password protocol.

Notation:

- Since each key generated by the principal is different from all the ones that were previously generated (Assumption 1), the keys form a *totally ordered set*

  $$K = \{K_0, K_1, K2, ..\}$$

  whereby K_i is generated before K_j if $i < j$ or, equivalently, K_i is the last key generated before K_{i+1}.

- U_i is the *stable state* of U characterized by the knowledge of K_i as the current key.

- U_i' is the *transient state* of U corresponding to the transition from state U_i to state U_{i+1}.

- AS_i is the state of AS characterized by the knowledge of K_i as the current key representing the principal.

- !m represents the sending of message m.

- ?m represents the receipt of message m.

- := denotes the assignment operation.

- K_u is the state variable of U that represents the current key.

- K_{as} is the state variable of AS that represents the current key.

- $cpw(K_i)$ is the command entered by the principal that triggers a password change operation on U. The old password is K_i and the new one is K_{i+1}.

- m_i is the REQ_CPW message where $K_{old} = K_i$ and $K_{new} = K_{i+1}$

- $\bar{m}_i$ is the REQ_CPW message, a retransmission from the AS's point of view, where $K_{new} = K_{i+1}$ and K_{old} can take any value.

- r_i is the positive (*accept*) response to m_i.

6.3 Properties of the idealized Protocol

Due to the special structure of the protocol messages the protocol has the following properties:

Property 1
AS_i can correctly determine the transition (the loop or the transition to the next state) to be executed because AS_i can always tell apart m_i from m_{i+1}. When AS_i receives a message m, it can identify the message as m_i if $K_{new} = K_{as}$ and as m_{i+1} if $K_{old} = K_{as}$. This property is based on Assumption 1 and on the structure of REQ_CPW's which distinguishes the "new" key from the "old" one. In other words, K_{new} can never be mistaken for K_{old} and vice versa.

Property 2:
With the knowledge of K_i, AS_i can always extract the new key K_{i+1} (K_{new}) contained in message m_{i+1}.

6.4 Analysis

Goal:
Now we can show that, using this protocol, the principal and the AS will always (eventually) agree on the same key and that there will never be a deadlock situation even in the presence of message losses and replay attacks.

Basic Idea :
We will first show the single step correctness of the protocol, that is, if the principal and the AS both are in a state where both "know" the same key, and the principal triggers the password change protocol, then the following events take place:

- the protocol terminates after a finite period of time.
- after the termination of the protocol, both the principal and the AS will have replaced the old key with the new key provided by the principal.

Initial Equilibrium:
A complete proof of the protocol correctness requires that the system start in a *good* initial state, i.e., the combination of the states of the principal and the AS when both are *initialized* with the same key.

Single step correctness:
Let's assume that U is in state U_i, that AS in state AS_i, that they both consider K_i as the current key and that the principal enters cpw(K_{i+1}),

- **the protocol terminates:** because of Assumption 2 there will eventually be a pair of messages m_{i+1} and r_{i+1} that will be received (by AS and principal, respectively) after a finite number of retransmissions. After protocol termination U will reach state U_{i+1} and AS the state AS_{i+1}. Any transition of U to a state other than U_{i+1} and any transition of AS to a state other than AS_{i+1} are excluded because of Property 1.
- **after protocol termination U and AS will have replaced the old key with the key provided by the principal:** The value of K_u in state U_{i+1} can only be K_{i+1} since this is assured by the good behaviour of the program independently of the communication between U and AS. By definition m_{i+1} contains K_{i+1} in the position of the new key and by virtue of property 2 AS_i can extract K_{i+1} from m_{i+1} and assign its value to K_{as}. Thus AS also knows K_{i+1} in state AS_{i+1}.

The correctness proof is still valid when we take into account the effect of a *crash* on the side of either the principal or the AS. If the principal's end-system crashes in a transient state U_i' between state U_i and U_{i+1} the principal needs only to restart the protocol by entering cpw(K_{i+1}) ($K_{old} = K_i$ and $K_{new} = K_{i+1}$) in order to properly terminate the single step execution of the protocol. If the AS crashes, assuming that its non-volatile memory is crash-proof (otherwise there would be no recovery at all), and if the duration of the crash and recovery is comparable to a period of retransmissions from the principal's point of view, the crash has the same effect as the loss of a message and the protocol has been shown correct in the presence of failures and message losses. If the crash and recovery takes a very long time, the change password command will abort and the principal must restart from state U_i as in the case of the crash at the principal's side, but this is still considered to be only *one* protocol run and the principal must stick to $K_{new} = K_{i+1}$ since the server might have updated the password database before the crash.

7 Summary

The proposed protocol is based on a single and atomic challenge/response exchange. The possibility of atomic re-tries provides for a level of robustness and security that is not possible with current protocols. The advantages of the proposed key change mechanism can be summarized as follows:

1. The protocol is resistant to replay attacks due to the asymmetric flip-flop property of the token construction in REQ_CPW.

2. Unlike traditional authentication protocols, the messages in CPW do not need to contain an explicit challenge (timestamp or nonce) to demonstrate their freshness and to counter replay attacks. This property is a result of the logical temporal order provided by the sequence of keys and the robustness of the protocol messages. Because of the inherent strong synchronization on keys that the protocol provides, replay detection can be based on the sequencing of keys.

3. The protocol offers protection against walk-by-attacks whereby a previously authenticated principal leaves the workstation unattended.

4. Optionally, the protocol can permit recycling of keys without being susceptible to replay. By using a timestamp as one of the "nonces", the key sequence can be roughly anchored in time without requiring secure time-services. This is because the timestamp has to be *fresh* only with respect to the relatively low frequency of password changes.

5. The protocol can be resumed after a system crash on the side of either the AS or the requesting principal.

6. Last, but not least, the protocol is compact both in number of messages and individual message sizes, and it requires very few cryptographic computations.

References

[1] National Bureau of Standards, *Federal Information Processing Standards*, National Bureau of Standards, Publication 46, 1977.

[2] J. G. Steiner, B. C. Neuman, J. I. Schiller, *Kerberos: An authentication service for open network systems*, Usenix Conference Proceedings, Dallas, Texas, pp. 191-202, February 1988.

[3] Open Software Foundation, *DCE User's Reference Manual*, Cambridge, Massachusetts, 1992.

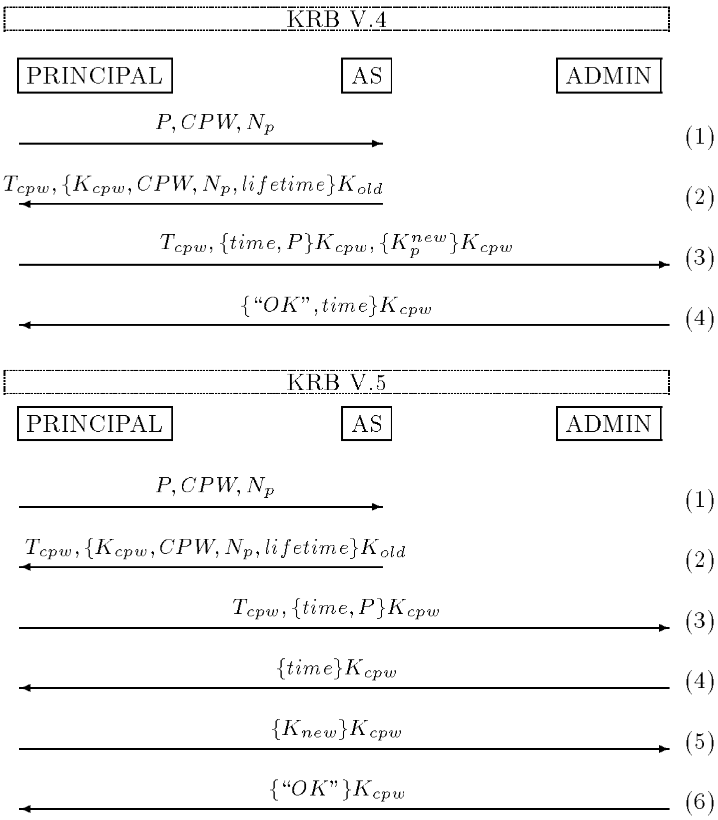
[4] S. M. Bellovin, M. Merrit, *Limitations of the Kerberos Authentication System*, Computer Communication Review, vol. 20(5), pp. 119-132, October 1990.

[5] G. Tsudik, E. Van Herreweghen, *On Simple and Secure Key Distribution*, Proceedings of 1993 ACM Conference on Computer and Communications Security, November 1993.

[6] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, M. Yung, *Systematic Design of a Family of Attack-Resistant Authentication Protocols* , IEEE JSAC Special Issue on Secure Communications, July 1993.

[7] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, M. Yung, *A Modular Family of Secure Protocols for Authentication and Key Distribution* (DRAFT) in submission to IEEE Transactions on Communications, August 1993.

[8] R. Molva, G. Tsudik, E. Van Herreweghen, S. Zatti, *KryptoKnight Authentication and Key Distribution Service*, Proceedings of ESORICS 92, October 1992.



Legend:

ADMIN	administration server (separate process co-located with the AS)
P	principal's name
N_p	nonce issued by principal
K_{cpw}	session key (between principal and ADMIN)
K_{old}	old key or password shared between principal and AS
K_{new}	new key or password to be shared between principal and AS
T_{cpw}	$\{K_{cpw}, P, lifetime\}K_{admin}$
$\{Y\}X$	message Y encrypted under key X. Before encryption, a sequence number as well as a checksum are added to the message, to prevent undetected replay or corruption of the message.

Figure 1: The Kerberos CPW protocols

PRINCIPAL | AS

$$\text{REQ\_CPW} = \{N1, N2, P, T(K_{new})K_{old}, T(K_{old})K_{new}\} \qquad (1)$$

$$\text{REP\_CPW} = MAC\{accept, REQ\_CPW\}K_{old} \qquad (2)$$

$$\text{or REP\_CPW} = MAC\{reject, REQ\_CPW\}K_{old}$$

$$\mathrm{T(K_{new})K_{old}} = \mathrm{MAC\{N1, N2, P, AS\}K_{old}} \bigoplus \mathrm{K_{new}}$$

$$\mathrm{T(K_{old})K_{new}} = \mathrm{MAC\{N1, g(N2), P, AS\}K_{new}} \bigoplus \mathrm{K_{old}}$$

Legend:

MAC{Y}X	DES-like Message Authentication Code (MAC) of Y with key X
$X \oplus Y$	exclusive-OR (XOR)
N1, N2	nonces
AS	name of the AS
T(X)Y	ticket containing key X sealed with key Y

Figure 2: Secure and Fault-Tolerant CPW protocol

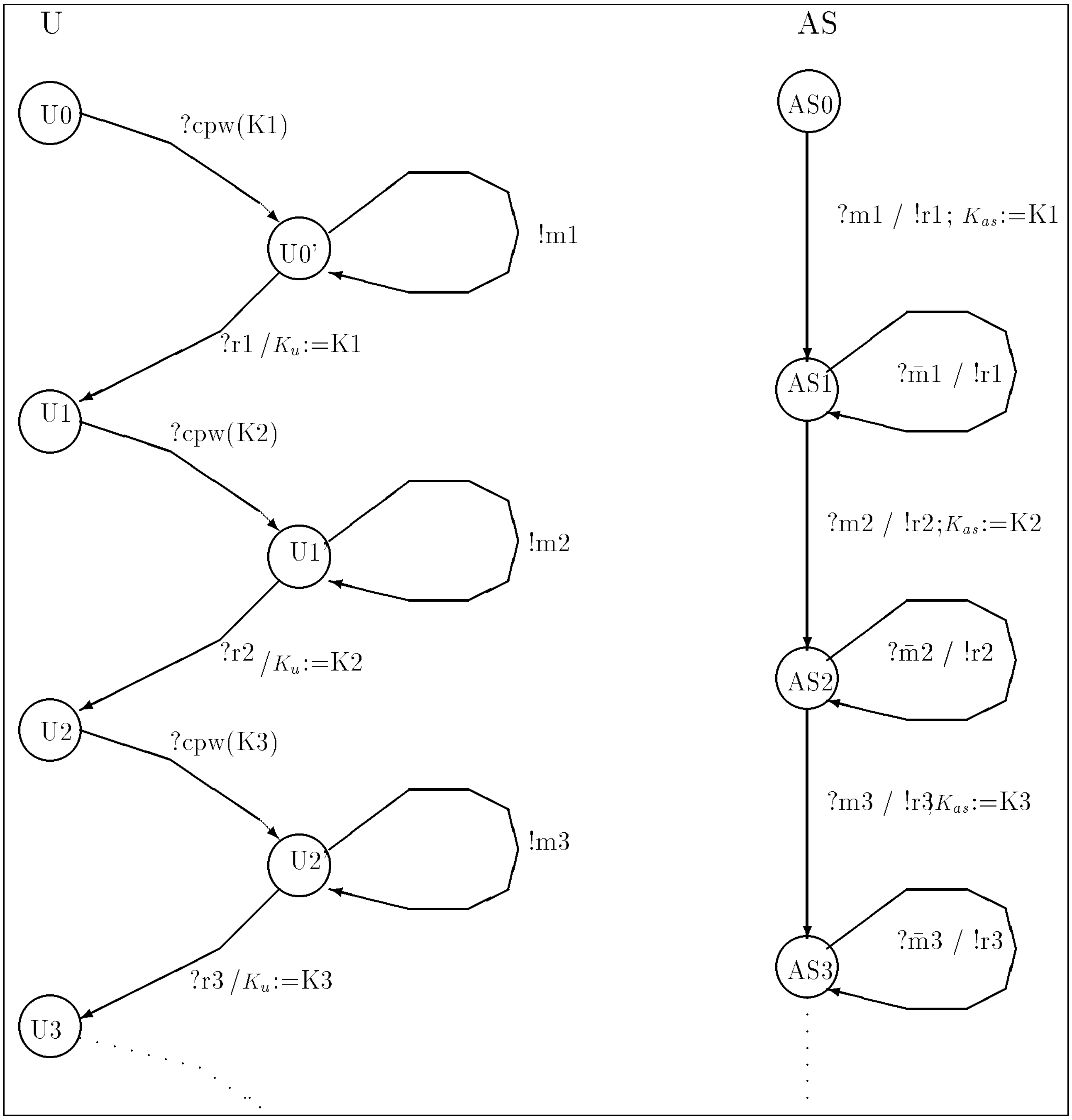


Figure 3: Idealized CPW protocol