Systematic Design of a Family of Attack-Resistant Authentication Protocols

September 1992

R. Bird
I. Gopal
A. Herzberg
P. Janson
S. Kutten
R. Molva
M. Yung

IBM Raleigh, Watson & Zurich Laboratories

Systematic Design of a Family of Attack-Resistant Authentication Protocols

R. Bird[1]
I. Gopal[2]
A. Herzberg[2]
P. Janson[3]
S. Kutten[2]
R. Molva[4]
M. Yung[2]

Abstract

The extensive use of open networks and distributed systems poses serious threats to the security of end-to-end communications and network components themselves. A necessary foundation for securing a network is the ability to reliably authenticate communication partners and other network entities. One-way, password-based authentication techniques are not sufficient to cope with the issues at hand. Modern designs rely on two-way, cryptographic authentication protocols. However, most existing designs suffer from one or more limitations: they require synchronization of local clocks, they are subject to export restrictions because of the way they use cryptographic functions, they are not amenable to use in lower layers of network protocols because of the size and complexity of messages they use, etc. Designing suitable cryptographic protocols that cater to large and dynamic network communities but do not suffer from the above problems presents substantial challenges in terms of ease of use, efficiency, flexibility, and above all security.

This paper discusses the above challenges, shows how a few simple protocols, including one proposed by ISO, can easily be broken, and derives a series of desirable properties that authentication protocols should exhibit to meet the requirements of future large and dynamic network communities. Then the paper describes a methodology that was developed to systematically build and test the security of a family of cryptographic two-way authentication protocols that are as simple as possible yet resistant to a wide class of attacks, efficient, easy to implement and use, and amenable to many different networking environments. It also discusses several concrete examples of protocols of that family that present various advantages in specific distributed system scenarios.

1. Introduction

The extensive use of open networks and distributed systems poses serious threats to the security of end-to-end communications and network components themselves. A necessary foundation for securing a network is the ability to reliably authenticate communication partners and other network entities. One-way, password-based authentication techniques are not sufficient to cope with the issues at hand. Modern designs rely on two-way, cryptographic authentication protocols. However, most existing designs suffer from one or more limitations: they require synchronization of local clocks, they are subject to export restrictions because of the way they use cryptographic functions, they are not amenable to use in lower layers of network protocols because of the size and complexity of

[1] IBM Corp., Research Triangle Park, NC.

[2] IBM T. J. Watson Research Center, NY.

[3] IBM Zurich Research Laboratory, Switzerland.

[4] Institut Eurecom, Sophia-Antipolis, France.

messages they use, etc. Designing suitable cryptographic protocols that cater to large and dynamic network communities but do not suffer from the above problems presents substantial challenges in terms of ease of use, efficiency, flexibility, and above all security.

1.1. Scenario and Attacks

Consider two parties A and B that share a key for use with some 'secure' cryptographic system. Using that key, the two parties may execute over time many instances of some agreed upon cryptographic authentication protocol, where each instance is independent of the others. Several executions may even be carried out in parallel if the two parties want to interact over multiple parallel authenticated connections or communication instances. Whenever one of the parties completes an execution of the protocol, it marks the execution as either accepted (in case of successful authentication) or rejected. The intention is that executions marked as accepted correspond to runs of the protocol that definitely involved the intended other party. An error-free history of the protocol runs between A and B is one in which all of the executions accepted by both parties (i.e. excluding those rejected and those possibly not yet completed because messages are still in transit) match exactly one-to-one.

An attacker of such a protocol can be intuitively described as a third party who has no access to the key. However, it has access to all legitimate protocol executions that were accepted by A and B in the past, and it is able at any time to interfere with on-going or to start new (faked) protocol executions involving A and/or B. The attacker's objective is to cause A or B to erroneously mark one of these perverted protocol executions as accepted, even though it does not match with any execution accepted by the other party.

Note that the attacker may adaptively send any message to either party, initiate new executions of the protocol with either party, and intercept messages sent by either party. We do not impose response-time constraints on these actions. Indeed, in real networks, two parties that share a key may often initiate multiple protocol executions simultaneously, and delays and message losses are normal events.

It follows that a third party can always act as a simple delaying 'relay' between the two parties without being considered an attacker. Indeed, such delaying is no different and is indistinguishable from what any network switch does in normal operation. In particular, this does not contradict our definition of an accepted protocol instance since both parties are indeed involved and both accept the exchange. In that respect it is the exchange that is authenticated and not the parties through which it is relayed. The third party plays only a passive role with no observable effect to A and B. Even though we will refer to this scenario as a **relay attack**, it is not a real attack in the sense that it should be prevented. In fact, no authentication protocol, however sophisticated, can prevent this 'relay attack' as it corresponds to a scenario that occurs all the time in the normal course of things. Even a third party that removes messages on links between A and B only creates a longer delay in the execution of the protocol, but the authentication will eventually either be rejected (e.g. following a time-out) or be accepted by both parties; the third party cannot conduct a successful attack merely through delaying or dropping messages.

The design of a correct and secure authentication protocol should take into account all the above considerations, as well as protect against traditional cryptographic weaknesses of course.

1.2. Related Work and Existing Designs

Many designs dealing with authentication in networks or distributed systems combine the issues of authentication with those of key distribution. These designs typically assume that all network parties share a key with a common trusted entity, often called a key distribution center (KDC), from which they can get pair-wise shared keys to carry out mutual authentication protocols. These protocols are called three party authentication protocols, and have been studied extensively e.g. Needham78, Denning81, Bauer83, Otway87, Burrows89, Steiner88, Bellovin90, ISO8732, Stubblebine92“. Most of the corresponding implementations e.g. Steiner88, ISO8732“ require the exchange of long messages, which is no problem for application-layer protocols, but makes them unsuitable for use in lower-layer networking protocols where limited packet sizes are an important consideration. Some require synchronized clocks e.g. Steiner88“ or counters e.g. ISO8732“ that pose system management and initialization issues, as will be discussed soon.

Two-party authentication protocols received less attention in the literature, despite their use in many networks. Some of these use public-key cryptography e.g. Jueneman85, Meyer82, ISO9594, Ianson90". With a public-key cryptographic system, each party only has to know and verify the public key of the other party, and there is no need to share secret keys. The ISO9594" protocol also requires large messages that make it unsuitable for use in low-level network protocols. The protocols we discuss in this paper assume shared-key cryptography because it is much more efficient, which is an important consideration for low-level network protocols. However they could also be operated with public key cryptography. The only other published protocol that we are aware of which uses shared keys is the proposed ISO standard ISOSC27" whose weakness is discussed in this paper.

1.3. Outline

This paper elaborates on the design of a family of secure authentication protocols that was originally discussed in Bird91". In a first part, starting from the basic principles of cryptographic authentication and looking at existing designs, we show that several simple protocols, including one that was discussed by ISO, are easily broken. Through an examination of the problems encountered in such designs, we derive a set of requirements for defining cryptographic authentication protocols that are at the same time simple and yet resistant to a wide class of attacks, while being easy to use and applicable to many networking environments.

Based on the requirements outlined in the first part, the second part of the paper then develops systematically a canonical form of a simple cryptographic two-way authentication protocol. We show that protocols matching this canonical form resist a wide range of attacks, we show a couple of concrete embodiments of the canonical form, and we discuss the properties of such protocols. One of the simple embodiments of the canonical form has been proven secure against any attack under reasonable cryptographic assumptions Bird91". However this proof extends beyond the scope of the present paper.

2. Basic Principles and Possible Attacks

2.1. Basic Principles of Authentication

Passwords

The authentication method most widely used in network environments today consists of asking users to prove their identity by demonstrating knowledge of a secret they know, typically a password. This widespread but old technique has several weaknesses:

Passwords are transmitted in the clear: In most password systems, the password typed by a user is sent in cleartext over the network to the computer that requested it. This means that an intruder equipped with suitable electro-magnetic tools can tap network lines and spy on passing traffic to collect passwords.

Passwords are easy to guess: A relatively low fraction of potential intruders are so dedicated and affluent that they would actually resort to the above electronic wire-tapping. However simpler attacks are also possible. Since users need to memorize their passwords, they typically select passwords that they can easily remember. Unfortunately, experience has shown that such passwords are selected from within a relatively small vocabulary, and are thus easily guessed by potential intruders Morris79, Spafford89".

Authentication is one-way only: Last but not least, password schemes are typically used for one-way authentication only, i.e. computers ask users for their passwords but users never question that they are communicating with the right computer, and thus never challenge a computer to provide any password. This is however a serious exposure: when a user sits at a workstation and requests services from some remote computer, there is no proof that a password prompt appearing on the screen is from the right

computer; the prompt could be a faked one displayed by another computer or by a corrupted application designed by an intruder.

Other Techniques

There exist many alternative techniques to password-based authentication, where users do not have to remember anything so that the risk of an intruder guessing a password is inexistent. For instance, some systems base authentication on recognition of biometric information such as a voice sample, a finger-print, or a hand signature. However, such authentication techniques have reliability problems and require relatively expensive hardware support, so that they are found only in selected high-security environments, not in general-purpose network environments. Besides, they still suffer from the problem that the biometric information is often transmitted in cleartext over network lines that can be tapped, thus allowing replay at a later time.

Of all authentication techniques that circumvent the drawbacks of passwords, the most promising ones are those using cryptographic means, whose common use is becoming increasingly feasible thanks to technological progress. With such techniques, users are typically provided with smart cards or chip cards equipped with a processor capable of cryptographic operations. Such cards offer means to communicate either with their owner (through a liquid-crystal display and possibly a numeric keypad) or directly with a computer (through an electronic interface). Whatever the interface details, authentication is based upon using the card to compute or verify cryptographic messages exchanged with the computer.

Cryptographic Techniques

The basic idea of cryptographic authentication consists of challenging the user or communicating party being authenticated to prove its identity by demonstrating ability to encipher or decipher some item with a secret or private key known to be stored inside the smart card (or other secure storage device if the communicating party is a computer).

Of course, since the secret or private key stored on the card or secure device changes infrequently, the item to be enciphered or deciphered with it must change for every execution of the authentication protocol, otherwise, even though the secret never flows in cleartext over the network, an intruder could still tap a line, record the cryptic message that flows over it, and play that recording back at a later time without even knowing what it means.

To guarantee that the item that gets enciphered (or deciphered), called the **challenge**, is different for every execution of the authentication protocol, three techniques are used. The challenge may be derived either from a real-time clock reading, in which case it is called a **time-stamp**, from a **counter** that is incremented for every protocol execution, or from a random number generator, in which case it is called a **nonce**. In any case, a new challenge (time-stamp, counter value or nonce) is generated for each protocol run.

With time-stamps, the user or entity being authenticated, further referred to as A, enciphers the current reading of its clock and sends the result to the computer or party requesting the authentication, further referred to as B. B then deciphers the received message, and verifies that the time-stamp corresponds to the current real-time. The drawback of time-stamps is thus immediately apparent. A and B must have synchronized real-time clocks for the verification to be possible. However, since clocks can never be perfectly synchronized and messages take time to travel across networks anyway, B cannot expect that the deciphered time-stamp received from A will ever be equal to its own real-time clock reading. A's time-stamp at best can (and normally should) be within some limited time window of B's real-time clock. However, as soon as a time window of tolerance is defined, a potential intruder could exploit it to impersonate A by replaying one of A's recent authentication messages within that time window. Preventing this requires putting a limit to the number of authentication protocol runs allowable during the time window, and having B save all the authentication messages within the window. Thus both efficiency and security require pretty good clock synchronization. Achieving such synchronization is often difficult, and making it secure would itself require using some other authentication method not depending on time.

With counting challenges, A and B maintain synchronized counters of the number of times they authenticated one another. Every time A wants to communicate with B, it enciphers its counter and sends the result to B, who

deciphers it and verifies that it matches its own counter value, whereupon both parties increment their respective counters for the next time. The drawback of such counting challenges is that both parties must maintain synchronized counters, which poses stable storage and counter management problems in the long run. The counters must be long enough to prevent an attacker from waiting for a (deterministic) counter wrap-around; using counters complicates resolution of conflicts when both parties want to initiate protocol executions at the same time; after detecting loss of synchronization between two counters, some other method of authentication must be used to securely re-synchronize them; and an undetected error in the counter values may have catastrophic effects.

We conclude that both time-stamps and counters techniques, while useful, are not a complete solution, especially for protocols in the lower layers of a network architecture. The best technique for this purpose, and also the simplest one to implement, consists of using nonces. The price for this simplicity is an extra network message. While A can authenticate itself to B with a single message if time-stamps or counters are used, two messages are necessary with nonces. Indeed, it is B and not A who needs to be sure that the nonce was never used before. Thus it is B and not A who must generate it. B must encipher it and send it to A, then A must decipher it and send it back in cleartext for B to authenticate it, which costs a total of two messages. The cost of one more message is usually tolerable, especially since the extra message can often be piggy-backed onto regular traffic or its cost can be amortized over many subsequent messages authenticated with sequence counters. Given these advantages of nonces over time-stamps and counters, the rest of this paper focuses on nonce-based authentication protocols.

2.2. Simple-minded Cryptographic Authentication

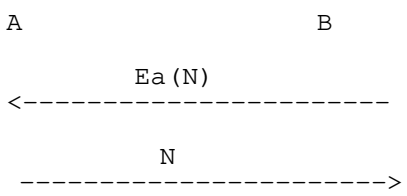

Figure 1. One-way authentication of A

Figure 1 is a simple representation of the nonce-based one-way authentication protocol described in the previous section, where N is a nonce generated by B and Ea(N) is its value enciphered under some key Ka that B knows is associated with A: it may be a secret key shared between A and B or the public key of A. The protocol allows B to authenticate A.

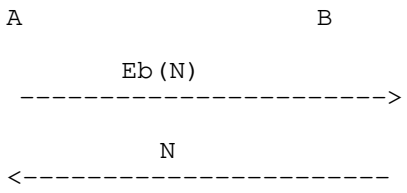

Figure 2. One-way authentication of B

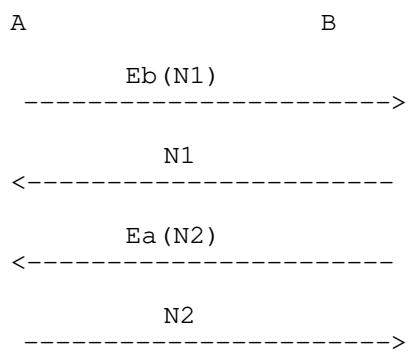

Figure 3. Two-way authentication

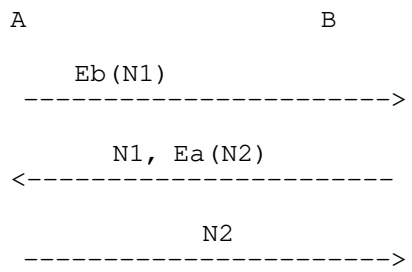

Figure 4. Two-way authentication with three messages

As suggested earlier, many scenarios in practice require two-way rather than one-way authentication, i.e. A must also authenticate B. This could of course be achieved simply by inverting the roles of A and B in Figure 1 on page 5, as represented in Figure 2 on page 5, where Eb(N) stands for the value of N enciphered under a key Kb that is either a secret key shared between A and B or the public key of B. Putting the two protocols together leads to Figure 3 for a complete two-way exchange. One can then immediately observe that the resulting protocol may be simplified by combining the second and third messages to save one network transmission, as represented in Figure 4. With symmetric cryptographic systems, the keys Ka and Kb could be the same, so that Ea and Eb indicate encryption under the same shared secret key.

Unfortunately, this simple and elegant two-way authentication protocol, which was derived most naturally by combining two instances of the simplest possible one-way protocol, presents a number of undesirable characteristics. In particular, we will show that an intruder may break this protocol by interleaving messages from different executions of it, if the same shared key is used by both parties **or** if the same key is always used by the party who starts the protocol. One could still argue that the protocol is safe if used with four different keys, however this creates additional overhead and key management issues, as well as migration problems for existing designs. In the next subsection we discuss the problems and the attacks, which will later help explain the issues associated with designing a simple and secure protocol using only one key and symmetric cryptography.

2.3. Attacks on Simple Cryptographic Authentication

The simple two-way authentication protocol illustrated in Figure 4 suffers from a number of defects:

Known plaintext attacks: A first weakness of the protocol, which is not a real exposure but still an undesirable feature, is its openness to known plaintext attacks. Every enciphered message flowing between A and B is the ciphertext of a bit string (the nonce) that flows in plaintext in a subsequent message between A and B. This enables a passive wire-tapping intruder to collect two cleartext-ciphertext pairs every time the protocol is run, which at the very least helps it accumulate encryption tables in the long run, and may even help it break the scheme and discover the encryption key, depending on the quality of the

encryption algorithm used. It is in general desirable that the plaintext of exchanged enciphered messages not be known or derivable by intruders.

Chosen ciphertext attacks The situation is in fact worse in the sense that a potential intruder may even turn known plaintext attacks into selected text attacks by playing an active instead of passive role. Pretending that it is A or B, an intruder may send the other party (B or A) a ciphertext message that it selected itself and wait for that other party to reply with the deciphered value of that text. Of course the intruder, not knowing the right key, may not be able to complete the third protocol flow. However, it can accumulate knowledge about cleartext-ciphertext pairs **of which it selected the ciphertext itself** (or the cleartext, if the challenge was enciphered instead of deciphered). So it may try specific ciphertext strings such as all zeros, all ones, or whatever else might help it break the key faster, depending on the cryptographic method used. It is in general desirable that an intruder not be able to trick legitimate parties into generating deciphered versions of selected ciphertext messages (or enciphered versions of selected cleartext) Biham90".

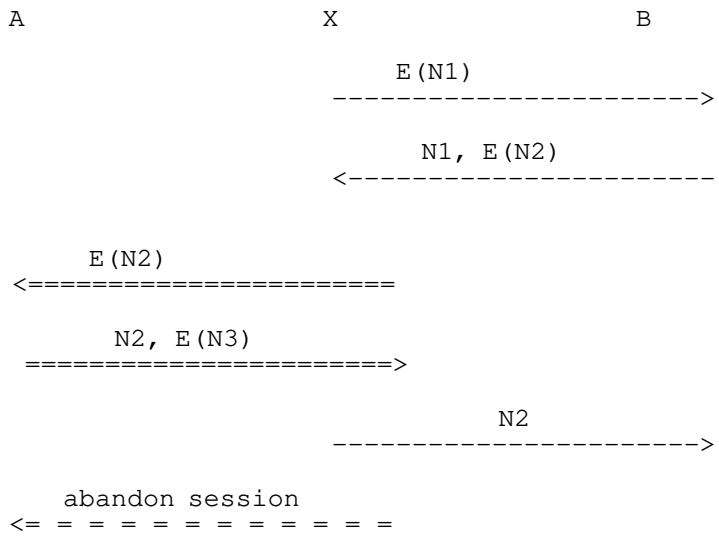

Figure 5. An oracle session attack

Oracle session attacks: In fact, if A and B use the same key in the simple protocol suggested earlier, the intruder actually **can** break the authentication without even breaking the key. This is illustrated in Figure 5, where the intruder, noted X and posing as A, starts a session with B by sending B some enciphered nonce E(N1). B replies with the deciphered value N1 and its own enciphered challenge E(N2). X cannot decipher N2, but it can take advantage of a selected ciphertext attack on A, using A as an oracle who will provide the necessary deciphered value N2. X accomplishes this by posing now as B to start a separate 'oracle' session with A, sending E(N2) as the initial message on that session. A will reply with the needed value N2 and some own enciphered nonce E(N3). X then drops the oracle session with A since it cannot and does not care about deciphering N3. But it then turns around, and successfully assumes its faked identity A with respect to B by sending E(N2) to B.

This example exposes one fundamental flaw of the protocol: the cryptographic messages used in each flow of the protocol must be different from one another in the sense that it must not be possible for an intruder to use messages appearing in the second flow to derive, reconstruct or fake messages needed for the third one.

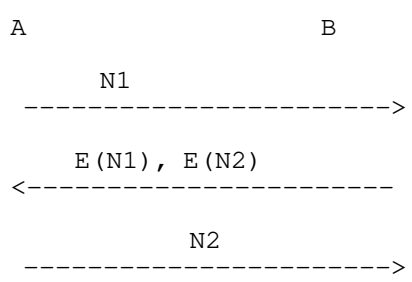

Figure 6. The ISO two-way authentication protocol

This observation was made independently by the designers of an improved authentication protocol defined by the ISO SC27 ISOSC27“ and represented in Figure 6. In this protocol, the challenge to the initiator of the protocol (A) consists of demonstrating ability to *en*cipher a given cleartext nonce, while the challenge to the responder (B) consists of demonstrating ability to *de*cipher a given ciphertext nonce. Thus an intruder can no longer misuse one party as an oracle 'decryption server' against the other. Unfortunately, this improved protocol still suffers from a fourth type of defect, also present in the original protocol, and discussed hereafter.

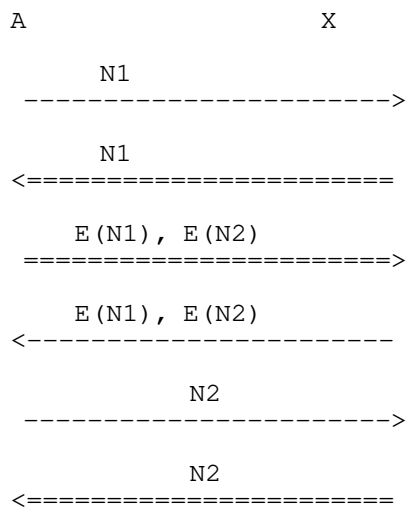

Figure 7. A parallel session attack

Parallel session attacks: Another defect commonly found in simple protocols such as those seen above is depicted in Figure 7, where the intruder assumes a passive role instead of an active one. It intercepts a call from A to B with challenge N1, and, leaving B completely out of the picture, turns A into an oracle against itself. Since X cannot answer the challenge N1 by replying E(N1), it simply pretends that it is B trying to start a parallel session with A. Of course it selects to use just the same N1 as the first challenge on the original session, thus causing A to provide it with precisely the answer E(N1) needed to complete the first authentication. A also sends its own enciphered challenge E(N2) on the parallel session. X then replies to the original challenge N1 by sending E(N1) back. With that reply, X may even send the same enciphered challenge E(N2) it just got from A. A will then complete the first authentication exchange by replying with N2, which is again exactly what X needs to complete the second authentication. Thus X has succeeded doubly in posing as B on both the original and the parallel sessions! (It could also have dropped one of these, of course.)

Many connection set-up protocols at different layers of different network architectures do not allow the simultaneous establishment of multiple parallel sessions. However, there also exist many networking environments where such parallel sessions are legal by design. Where they are allowed, it would be feasible to warn implementers to check that the first challenge received on one session is not a replay

of a challenge outstanding on another session. However leaving session security up to implementers rather than guaranteeing it by design is bad practice. Furthermore, in typical implementations, session parameters are instantiated independently for each session so that it is not possible for one session to check on parameters of other sessions.

Parallel session attacks illustrate another fundamental flaw of many simple authentication protocols: the cryptographic expression used in the second message must be asymmetric (i.e. direction-dependent) so that its value in a protocol run initiated by A cannot be used in a protocol run initiated by B.

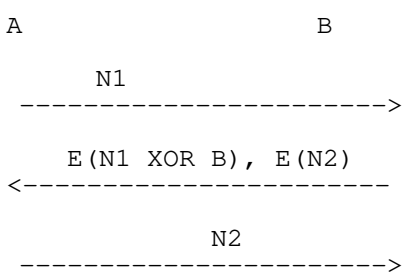

Figure 8. A more complex and asymmetric two-way authentication

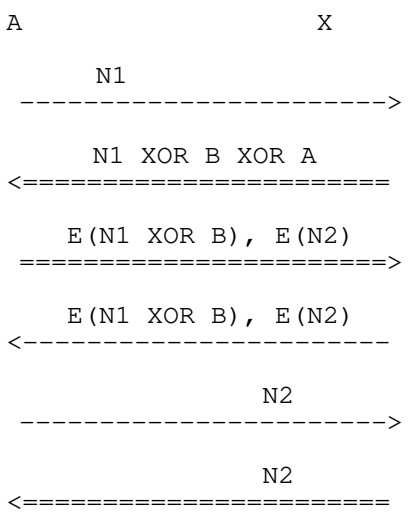

Figure 9. An offset attack (through a parallel session)

Based on the above observations, i.e. that the cryptographic message in the second flow must be asymmetric (direction-dependent) and different from the one in the third flow, one may be tempted to declare that a protocol such as the one in Figure 8 should be secure. Here, enciphering N1 has been replaced by enciphering a simple function of N1 (XOR stands for exclusive-or). Beyond the fact that known- and selected-text attacks are still possible, the problem now is that the simple function has not fixed anything: an intruder can still resort to a parallel session attack; it merely needs to use proper offsetting in addition, as illustrated in Figure 9.

The simple and apparently sound protocol of Figure 4 on page 6 is thus broken in many ways. In the course of our research, we have come across and designed many similarly simple and apparently secure two-way authentication protocols which, upon careful examination, proved to be subject to oracle-session attacks, parallel-session attacks, offset attacks, and/or other types or combinations of attacks that involved replaying messages and functions of challenges observed in other runs of the protocol. We further refer to all such attacks collectively as **interleaving** attacks. Thus we came to the conclusion that the design of simple protocols that resist such interleaving attacks is hard and requires a careful and systematic approach, as explained in the remainder of this paper.

2.4. Design Requirements

The objective of the work discussed here is thus to understand what it takes to design two-way authentication protocols that are complex enough to resist interleaving attacks, yet remain economical and simple enough to be usable in low layers of network architectures. To understand the boundary conditions on our design space, we summarize here requirements that such protocols must meet, most of which are derived from the discussion so far:

Nonce-based: Because of an explicit intention to avoid clock synchronization and stable counter storage issues, the protocols should preferably use nonces.

Resistant to common attacks: It must be possible to show that the resulting protocols are instrinsically secure against known- and selected-plaintext attacks, as well as interleaving attacks, i.e. that a potential intruder can never derive a 'good-looking' cryptographic message from any reply to or manipulation of past messages it may have observed or triggered itself.

Usable at any layer of any network architecture (small messages): The protocols must be suitable for low-level networking mechanisms as well as for application-layer authentication, meaning they must be based on cryptographic authentication messages that are as short as possible, hopefully the length of a single block of ciphertext with the given encryption algorithm.

Usable on any processing base (few computations): The protocols must be easily executable on smart cards as well as on low-end, entry-level networking components and workstations (e.g. PC's) with little processing power and no specialized cryptographic processing chip, i.e. they must involve as few cryptographic operations as possible.

Together, the latter two requirements, short and simple messages are aimed at securing communication in primitive environments, such as for instance a link-layer protocol or a secure boot service, where minimal complexity and minimal message size are required.

Using any cryptographic algorithm: The protocols must be able to use any of the known and typical cryptographic algorithms, either symmetric ones, such as DES DES77“, or asymmetric ones such as RSA RSA83“ for instance.

Exportable: Keeping in mind that import/export of cryptographic technology is tightly controlled in many countries, authentication protocols designed to be used without geographical restrictions should conform to applicable regulations. Most regulations restrict the import/export of technology allowing bulk data encryption/decryption. However, import/export of technology aimed solely at data integrity and authentication function is generally easier (subject to more easily acquired bulk licenses). Thus the chance to receive proper licensing for a technology is larger if it relies only on data integrity and authentication techniques and not on data confidentiality functions, i.e. if it provides only Message Authentication Code (MAC) functions and does not require full-fledged encryption and decryption of large messages. This constitutes an additional requirement for us, which is for instance not fulfilled by designs such as Kerberos or X9.17 that rely on encryption and decryption of large messages.

Extendible: The protocol should be flexible to accommodate different contexts and allow possible functional extensions, at least the most obvious ones. In particular, it should support the sharing of secret keys between multiple users (more than two) in scenarios where the savings compensate for the resulting decrease in security. Another obvious extension should allow carrying additional fields in the messages, which would thus be authenticated as part of the protocol.

Designing protocols with the above properties is hard and is the subject of the rest of this paper.

3. Systematic Derivation of a Minimal Protocol

Our objective in the rest of this paper is to construct systematically two-way authentication protocols meeting the security and other design requirements discussed above. To this end we first derive a canonical form for such protocols through a set of intuitive steps. Then we show how this canonical form can be tested systematically for security against interleaving attacks. These tests show necessary conditions for the security of the canonical protocol. Finally, we give examples of some of the simplest possible protocols realizing the canonical form and we discuss their relative merits. We proved in Bird91“ that one of the simple embodiments of the canonical form is secure against any attack, under reasonable cryptographic assumptions. However, this proof extends beyond the scope of the present paper.

3.1. Canonical Protocol

To produce a canonical form for all protocols fulfilling the security and design requirements seen above, we examine one-by-one each requirement and derive immediate implications on the possible canonical form.

Resistance to Replay Attacks through Use of Nonces

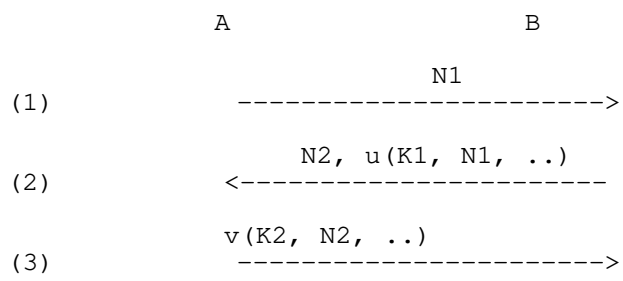

Figure 10. Canonical protocol 1: Resistance to replay attacks through use of nonces

Given that we focus on authentication protocols using nonces, the most general canonical form for all such protocols is depicted in Figure 10. Here, the initiator A sends some nonce N1 to the intended partner B. B authenticates itself by sending back a function u() of several parameters including at least its secret K1 and the challenge N1 it received from A. In turn it sends a challenge N2 to A. A finally completes the protocol by authenticating itself with a function v() of several parameters including at least its secret K2 and the challenge N2 from B.

Resistance to Oracle Session Attacks

As indicated in the earlier section on attacks, in order for this protocol to resist oracle attacks, the functions u() and v() must be different from one another, meaning that a cryptographic message from flow (2) can never be used to derive the necessary message for flow (3).

No Restriction on Cryptographic System

The protocols being developed must be workable with either symmetric or asymmetric cryptographic systems. With the latter, the secrets K1 and K2 would be the private keys of B and A respectively. However, with symmetric systems, K1 and K2 would be shared secret keys. In fact, in many typical scenarios, they would be the same shared secret key. We will pursue our step-wise protocol construction under that assumption because it opens the door to attacks that would not be possible otherwise, yet it has advantages of efficiency and ease of migration of existing systems over solutions which use different keys for A and B. Any protocol that resists interleaving attacks using equal symmetric keys is also safe with asymmetric ones while the opposite is not true.

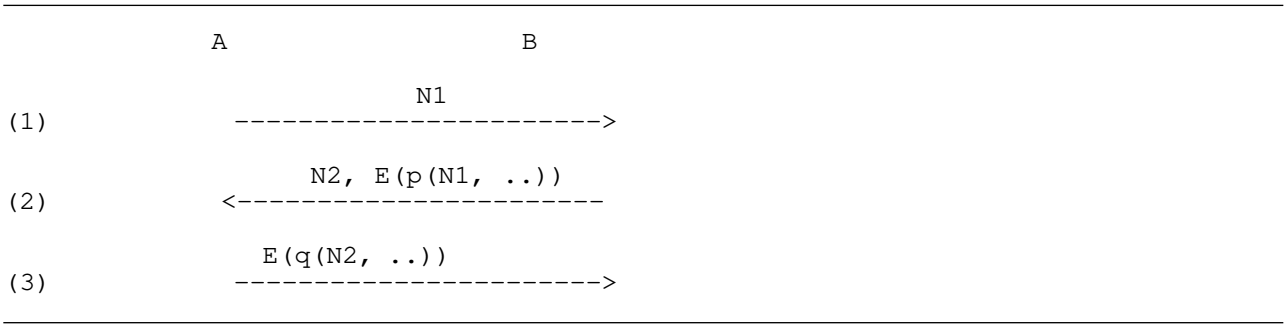

Figure 11. Canonical protocol 2: Use of symmetric cryptography

Under the assumption of symmetric cryptography with a single shared secret key K, our initial canonical form becomes as depicted in Figure 11, where the functions u(K1, ..) and v(K2, ..) become symmetric encryption (or decryption) operations (noted E) under the same key K of two different functions p() and q() of the remaining parameters.

Resistance to Parallel Session Attacks

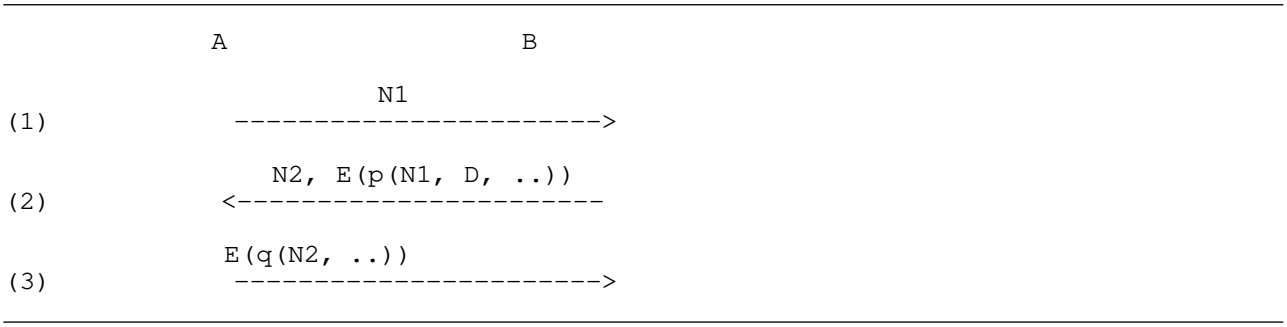

Figure 12. Canonical protocol 3: Resistance to parallel session attacks

As indicated in the earlier section on attacks, the prevention of parallel session attacks suggests that function p() must be asymmetric, i.e. direction-dependent. In other words, the arguments to p() must be different depending on whether A or B starts the communication session. This is depicted in Figure 12 by the addition of a parameter D in p(), which stands for anything indicating or tied to the direction of the flow (e.g. the name or address of one of the parties).

Small Cryptographic Messages

As shown in Figure 12, the complete protocol requires three messages. In any networking environment, these messages need not travel in packets of their own. In practice, they can and typically would be piggy-backed onto other network packets, such as, for instance, connection requests and confirmations at whatever layer entities are being authenticated (e.g. link, network, transport, application, etc.).

Within packets at any such layer, the authentication fields should take as little space as possible. As it stands presently, the canonical protocol requires carrying a nonce on the first flow, a nonce and a cryptographic expression on the second one, and a cryptographic expression on the third one. (The cleartext of the directional parameter, e.g. party name or address, is implicitly known or already present in the enclosing packet headers.) The required size of nonces is given by the amount of security desired: the larger they are, the lower the probability will be that a given nonce gets reused. The size of the cryptographic expressions depends on the encryption algorithm used and on the amount of information being encrypted.

To take a concrete example, if the DES is used, the smallest cryptographic expression will be 64-bit long. Since the security of the whole authentication protocol rests on the unforgeability of the cryptographic expressions, if expressions of 64 bits are used, the statistical security of the expressions is 2**64, the inverse of the probability that an intruder can successfully guess a cryptographic expression. For many environments such a degree of security is already quite satisfactory.

Without at all precluding that there exist environments where longer cryptographic expressions may be desired, we pursue our step-by-step derivation of a canonical form under the assumption that a degree of security of 2**64 is sufficient in most scenarios. This suggests that using 64-bit nonces is sufficient as well. Furthermore it suggests that p() and q() may be restricted to 64-bit functions of their parameters without compromising the degree of security. We further make this simplifying assumption. Using longer cryptographic expressions with standard DES encryption would add nothing since the probability of guessing a DES key remains anyway 1/2**64 (or more accurately 1/2**56).

Resistance to Offset and Selected-Text Attacks

The observation that it is desirable to restrict p() and q() to 64-bit functions to limit message size has severe implications. Indeed, many simple and attractive 64-bit functions of only N1 and D (e.g. XOR, rotations, conditional permutations, etc.) could be subjected to selected-plaintext and offset attacks by an intruder playing number games on its nonces to get what it wants enciphered by a legitimate party. As observed in the earlier discussion on possible attacks, it is important that the protocol resist such offset and selected-text attacks. Thus, there must be further conditions on the nature of p().

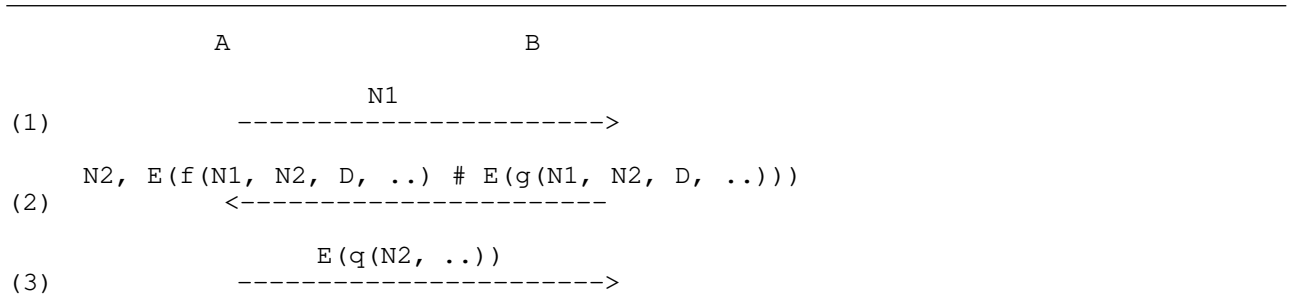

Figure 13. Canonical protocol 4: Resistance to selected-text and offset attacks

By including inside function p() an additional internal encrypted function of the same parameters N1 and D, one can separate these parameters cryptographically and thus bar the way to offset attacks. Further, by including N2 inside function p(), one bars the way to a potential intruder using B as an oracle to get some selected plaintext enciphered in flow (2). Indeed the cleartext of that flow then depends on nonce N2 which is not under the control of the intruder initiating a protocol run. These further conditions on p() are represented in the revised canonical form of Figure 13. In its most general form, p() would thus be a 64-bit function (noted #) of two operands, which themselves include functions f() and g() of N1, N2, and D.

The different fields in the message should be cryptographically separated, i.e. the attacker should not be able to control one field through another field. This is necessary to support the security function of each field, e.g. to prevent offset attacks as described before. This separation should be ensured by an appropriate selection of the functions f() and g(). We will later give exact conditions on f() and g() to prevent interleaving attacks, and give specific examples of 'good' f() and g().

Few Cryptographic Operations

Any peer authentication protocol requires at least two cryptographic expressions (one in each direction), i.e. two cryptographic block operations if we assume the 64-bit size limitation as desirable and satisfactory for many scenarios. Figure 13 on page 13 suggests using a third cryptographic operation to protect against offset attacks. Yet minimizing the number of required cryptographic operations so the protocols remain efficient even on low-end processors is one of our requirements.

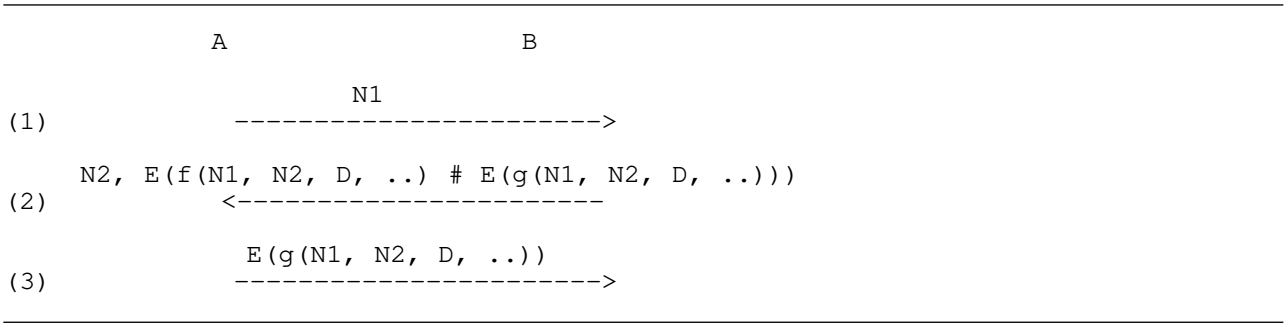

Figure 14. Canonical protocol 5: Minimal no. of encryptions

Fortunately, by putting additional conditions on q(), we can return to a protocol requiring only two cryptographic block operations. This can be achieved if desired by imposing that the functions g() and q() actually be the same, as represented in Figure 14. Indeed, in this case, the innermost cryptographic expression required to produce or verify flow (2) is the same as the cryptograhic expression of flow (3), and can thus be computed only once and saved to minimize computation.

Resistance to Interleaving Attacks

The step-wise derivation and refinement of more and more specific canonical protocols has lead us through a series of canonical forms where each one describes a subset of the all protocols described by the previous one. This series of canonical forms has taken into account a number of security features identified as requirements during our earlier interleaving attack analysis as well as additional requirements of simplicity, efficiency, and flexibility. However the derivation was purely intuitive. It remains to test that these protocols are indeed secure. In the next section we present the systematic method we used to test that they are secure against interleaving attacks. We have also shown formally in Bird91“ that one specific embodiment of the canonical form is secure against all attacks under reasonable cryptographic assumptions for the function E().

3.2. Testing Resistance to Interleaving Attacks

We present the systematic test of resistance against interleaving attacks based on the last canonical protocol presented above, in Figure 14. The test assumes cryptographic strength of the one-way function E(), and it assumes the intuitive but unproven notion that f() and g() are cryptographically separate in flow (2) of the canonical protocol. The test is however not a formal proof of security against all attacks, in the sense of the discussion presented in Bird91“. Still, it proved very useful for breaking many protocols that were suggested during our research, as well as the ISO protocol discussed above.

To limit the complexity of the equations used for the test, we assume that functions f() and g() take just N1, N2, and D as arguments. The possible existence of additional arguments only complicates the equations without adding anything to the test. However it is assumed that f() and g() are strongly dependent on each of their arguments in the sense that it is not possible for an intruder to manipulate any individual argument to cause f() or g() to take a predictable value independent of the other arguments. Further, we occasionally need to distinguish between the two

possible values of D, which we note as AB and BA, referring to the direction of the first and second flow respectively.

Our test method consists of showing that an intruder X attacking A or B cannot reconstruct or derive the cryptographic expressions needed in flows (2) or (3) by replaying or manipulating the results of passive observations or active interleaving attacks. We start by considering what information a potential intruder could possibly glean through passive observations or interleaving attacks in the course of time.

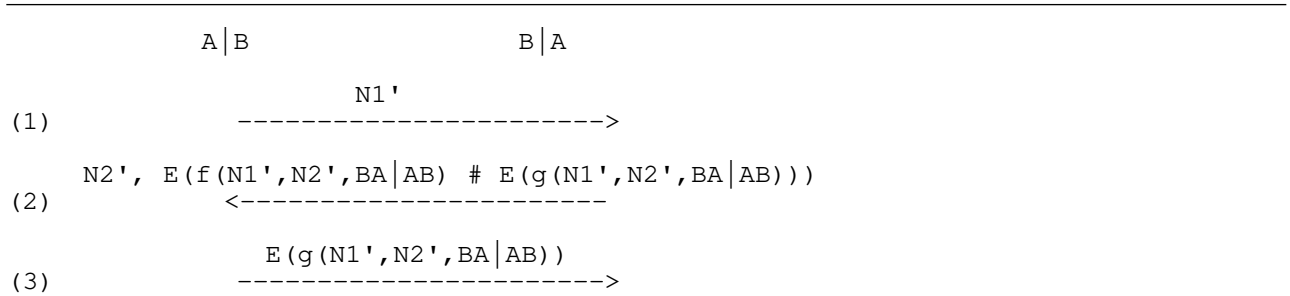

Figure 15. Reference session R1: a legitimate one

A first kind of observation an intruder can make consists of recording legitimate past runs of the protocol between A and B, such as the reference session 1 depicted in Figure 15. In this and subsequent figures, the use of the prime (') notation indicates that the variables are reference observations of the past.

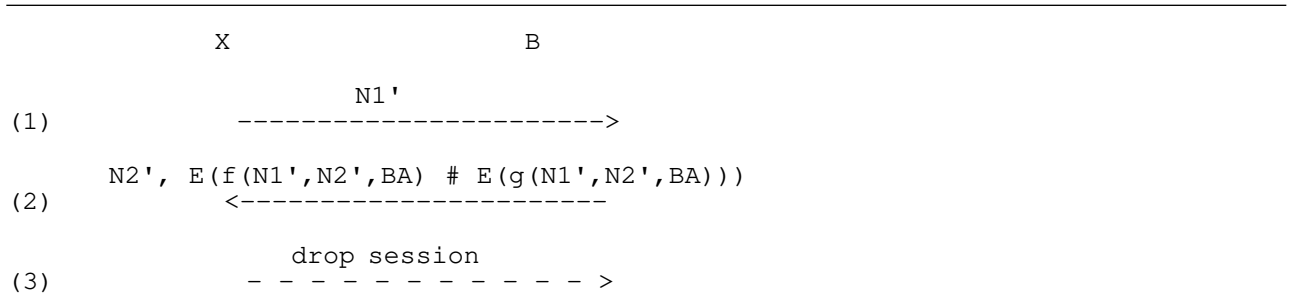

Figure 16. Reference session R2: X attacks B and fails

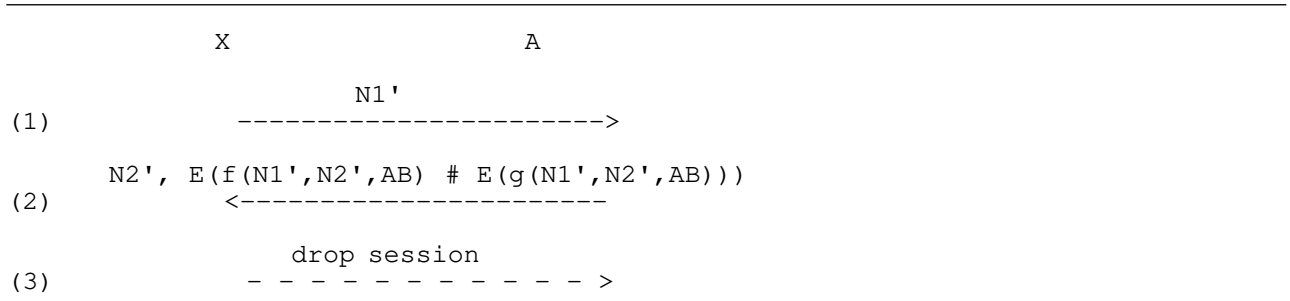

Figure 17. Reference session R3: X attacks A and fails

Similarly, X could collect information on the protocol actively by pretending to be A or B and attacking the other party (with no hope of succeeding for now) simply to see the other party's reaction and record more data on its responses. These two additional sources of information are represented as reference sessions 2 and 3 in Figure 16 and Figure 17.

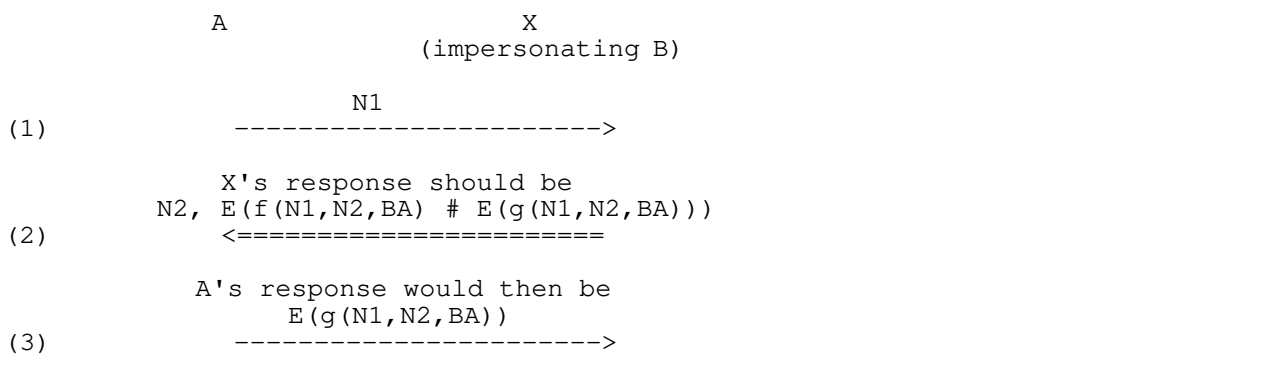

Figure 18. Attack session A1: X intercepts call from A (or B)

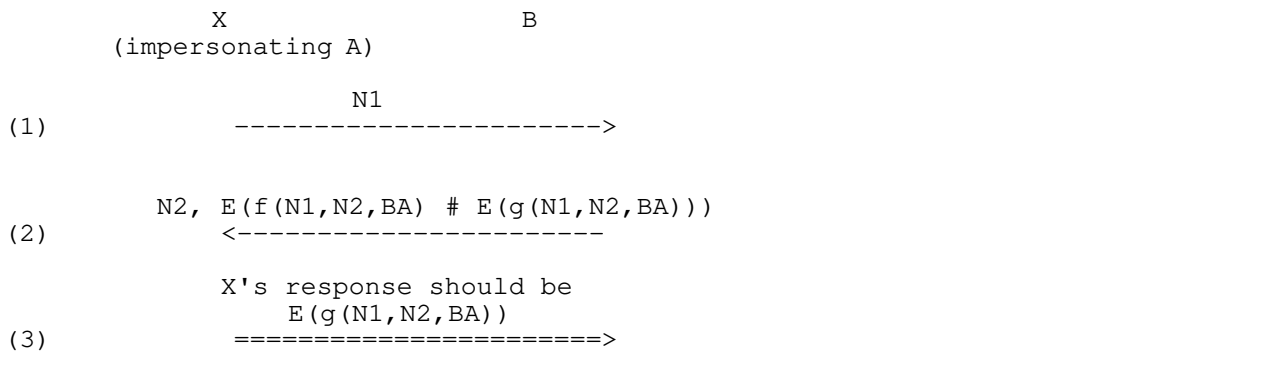

Figure 19. Attack session A2: X tries to calls B (or A)

Armed with any amount of knowledge about legitimate past runs or interleaved trial runs of the protocol, as defined above, an intruder could then try to attack the protocol in two ways represented in Figure 18 and Figure 19. In the first attack, X behaves passively, waiting for a call from A (or B, the derived equations would be similar) to intercept, while in the second it takes the initiative to call A (or B, the derived equations would also be similar). In the first attack, X is challenged to produce the cryptographic expression of flow (2), while in the second attack it is challenged to produce the one for flow (3).

In order to ensure security of the protocol against interleaving attacks, we must test that X cannot produce the necessary expressions based on interleaving attacks or simple manipulations of observations of the past, unless it was able to record the behavior of A and B for many possible combinations of N1' and N2', which we must assume is unrealistic. (If A and B communicated that often, they should have change secrets many times in the course of things!)

Since the protocols we investigate use challenges and responses that are always based on truly random numbers for every instance, combining information from various reference sessions in the hope that it may be useful for an attack is impossible (because the pieces of information from different sessions are 'independent' from one another). Thus, reference sessions are not combined, but rather used individually. The method, therefore, considers a single reference session for any attack session.

The test of security rests on the ability to show that the cryptographic expressions of flow (2) in the first attack (noted A1.2) and flow (3) in the second one (noted A2.3) can never be derived from any of the cryptographic expressions in flows (2) or (3) of any reference sessions (noted R1.2 and R1.3, R2.2, and R3.2) or of the on-going attack session itself.

Interleaving attacks do not involve exploiting weaknesses of E(). Namely, for any given message m the attacker can know E(m) only if it is able to cause the execution of the protocol by A or B to send a non-cryptographic one-to-one function of E(m). The attacker cannot produce E(m) itself because it does not know the right key.

Let us consider attack A1 first. To succeed in attack A1, the attacker should select N2, given some random N1, so that it knows E(f(N1,N2,BA) # E(g(N1,N2,BA))). In other words, it would have to be able to find the value of this expression in a past legitimate reference session R1, or to cause it to be sent in a reference session R2 or R3 by manipulating N1'. (Since the protocol always sends only encryptions, there is no hope to get a non-cryptographic function of the required value.) Furthermore, as explained in the Introduction, we don't consider the 'relay attack', where the attacker simply forwards N1 as N1' in reference session R2, as a real 'attack'. Therefore, the attacker must manipulate N2 so that it can obtain the value of E(f(N1,N2,BA) # E(g(N1,N2,BA))) from one of the flows R1.2, R1.3, R2.2 (but where N1' cannot be equal to N1 in R2.1) or R3.2.

In other words, for the attacker to succeed, one of the following equations should be satisfied:

(A1R1.2.a) f(N1,N2,BA) # E(g(N1,N2,BA)) = f(N1',N2',BA) # E(g(N1',N2',BA))
for random N1, N1' and N2'

(A1R1.2.b) f(N1,N2,BA) # E(g(N1,N2,BA)) = f(N1',N2',AB) # E(g(N1',N2',AB))
for random N1, N1' and N2'

(A1R1.3.a) f(N1,N2,BA) # E(g(N1,N2,BA)) = g(N1',N2',BA)
for random N1, N1' and N2'

(A1R1.3.b) f(N1,N2,BA) # E(g(N1,N2,BA)) = g(N1',N2',AB)
for random N1, N1' and N2'

(A1R2.2) f(N1,N2,BA) # E(g(N1,N2,BA)) = f(N1',N2',BA) # E(g(N1',N2',BA))
for random N1 and N2',
and N1' selectable by the intruder but different from N1

(A1R3.2) f(N1,N2,BA) # E(g(N1,N2,BA)) = f(N1',N2',AB) # E(g(N1',N2',AB))
for random N1 and N2', and N1' selectable by the intruder

We now consider these equations and derive sufficient conditions on f() and g() so they are insolvable without exploiting cryptographic weaknesses of E() (i.e. by an interleaving attack).

First consider (A1R1.2.a), (A1R1.2.b). These equations clearly would clearly hold with (cryptographically) only very low probability, if either f() or g() strongly depend on their first argument, since N1, N1, and N2' are random.

Consider now (A1R1.3.a) and (A1R1.3.b). Both equations also hold with only very low probability if g() depends strongly on its first argument. We conclude that this condition is good, since it also ensures that (A1R1.2.a,b) are not solvable.

Consider now (A1R2.2). Note that this would have been solvable by making N2 = N2' if N1' was permitted to be N1. However this corresponds to the uninteresting 'relay attack'. To prevent a solution with N1' different from N1, obviously we use the fact that g() strongly depends on its first argument. Since we do not consider cryptographic weaknesses, it is sufficient to prevent solution of both f(N1,N2,BA) = f(N1',N2',BA) and g(N1,N2,BA) = g(N1',N2',BA) together. One way of ensuring this is to make f() independent of N2.

Consider now (A1R3.2). This is solvable by N1'=N1 if g() and f() do not depend on their third argument, so we require dependency of at least one of them on the third argument. We also require that it be impossible to solve at once f(N1,N2,BA) = f(N1',N2',AB) and g(N1,N2,BA) = g(N1',N2',AB).

We now consider attack A2. Here, the attacker must manipulate the protocol to obtain E(g(N1,N2,BA)) from one of the reference flows. Thus one of the following equations should be satisfied:

(A2A2.2) g(N1,N2,BA) = f(N1, N2, BA) # E(g(N1, N2, BA))
in which the attacker tries to reuse the expression from the second flow of its attack for the third flow, such that N1 cannot depend on N2 since it is generated first

(A2R1.2.a) g(N1,N2,BA) = f(N1',N2',BA) # E(g(N1',N2',BA))
for random N2, N1' and N2'

(A2R1.2.b) g(N1,N2,BA) = f(N1',N2',AB) # E(g(N1',N2',AB))
for random N2, N1' and N2'

(A2R1.3.a) g(N1,N2,BA) = g(N1',N2',BA)
for random N2, N1' and N2'

(A2R1.3.b) g(N1,N2,BA) = g(N1',N2',AB)
for random N2, N1' and N2'

(A2R2.2) g(N1,N2,BA) = f(N1',N2',BA) # E(g(N1',N2',BA))
for random N2 and N2', and N1' selectable by the intruder

(A2R3.2) g(N1,N2,BA) = f(N1',N2',AB) # E(g(N1',N2',AB))
for random N2 and N2', and N1' selectable by the intruder

Consider first (A2A2.2). It is unsolvable (without a cryptographic attack) if either f() or g() strongly depend on their second argument. The same follows for (A2R1.2.a), (A2R1.2.b), (A2R2.2) and (A2R3.2).

Finally, consider (A2R1.3.a) and (A2R1.3.b). They could hold with only negligible probability, if g() strongly depend on its second argument.

To summarize, the following necessary requirements for f() and g() have been identified:

1. g() should strongly depend on its first and second argument, and either f() or g() should depend on the third argument (the direction). However, f() should not depend on its second argument.

2. It should be impossible to solve at once both f(N1,N2,BA) = f(N1',N2',AB) and g(N1,N2,BA) = g(N1',N2',AB).

Whether f() and g() functions meeting these criteria exist and can be found, and how good they are, are legitimate questions. The existence and quality of such functions are best substantiated by the examples given later.

3.3. Canonical Protocols with Additional Features

So far, we have developed a canonical protocol and tested its security against interleaving attacks. In particular, we have identified sufficient conditions on f() and g() to ensure that the protocol withstands interleaving attacks. In this subsection we continue to improve the canonical protocols, i.e. to give more detailed specifications for f() and g() while maintaining the security against interleaving attacks.

Resistance to Known Plaintext Attacks

As it now stands, the canonical form for our protocol is still open to known-plaintext attacks and can be refined and improved in that direction. Indeed, since all arguments of g() can be obtained (through wire-tapping) by a potential intruder and g() is not a secret function, that intruder can compute the cleartext and observe the ciphertext for flow (3). Similarly, since f() and all its parameters are no secrets, the intruder can compute f(). By combining it (with the # operation) to the ciphertext it observed for flow (3), it can derive the cleartext for flow (2), for which it can also observe the corresponding ciphertext on the network. Thus every run of the protocol still provides an intruder with two cleartext-ciphertext pairs, which is undesirable.

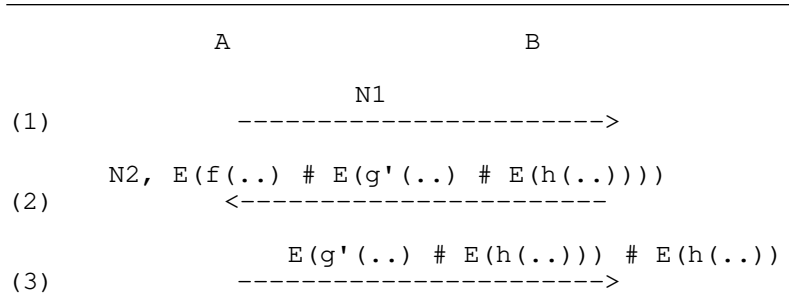

Figure 20. Canonical protocol 6: Resistance to known-plaintext attacks

Closing the door to such known-plaintext attacks does however require an additional encryption operation within g() so that the cleartext value for flow (3) becomes hidden from potential intruders: g() is replaced by g'() # E(h()), where # is again a suitable 64-bit operator. In addition, since the original ciphertext of flow (3) is part of the cleartext for flow (2), it too must be hidden, hopefully without requiring yet another encryption. This can be accomplished, as suggested in Figure 20, by using the third encryption not only inside g() but also outside, where it is combined to the original expression for flow (3) using some suitable 64-bit operator again noted by #. (f(), g'(), and h() in the latest canonical form all take the same parameters as f() and g() did in the previous form.)

In all fairness, it should be mentioned that this latest enhancement against known-plaintext attacks only increases the effort that an intruder would have to invest to carry out such attacks, but such attacks remain possible in a broader sense. Indeed, the intruder can find out all the cleartext inputs to the expressions of flows (2) and (3). Thus, while known-plaintext attacks are impossible in the strict sense of the word, similar attacks are always possible - though harder - if one regards these expressions simply as cryptographic functions u(..) and v(..) of known variables N1, N2, D, etc. Note, however, that the resulting plaintext space is much larger so that the added protection against known-plaintext attacks, while not absolute in the broad sense, tremendously complicates the task of a potential attacker at very low cost to the legitimate parties.

Exportability

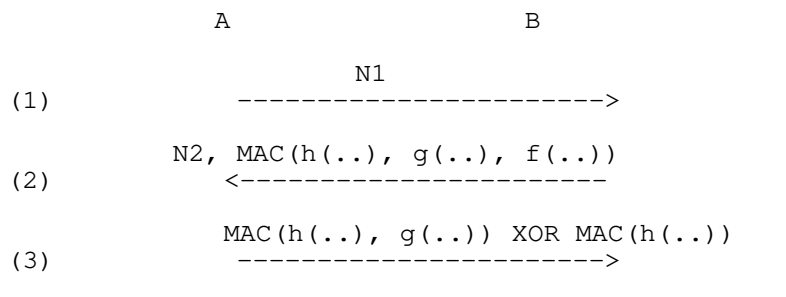

Figure 21. Canonical protocol 7: Exportability

With an eye towards exportability of authentication protocols, the best design is one that can be shown to use only one-way data integrity (MAC) operations instead of requiring both encryption and decryption operations.

In the specific case where the cryptographic algorithm used is DES, and all # operators in the latest canonical form of the protocols denote XOR operations, one observes that the resulting cryptographic expressions for flows (2) and (3) indeed boil down to combinations of plain 64-bit MAC integrity checks computed using the Cipher Block Chaining (CBC) mode of operation of DES, as represented in the canonical form of Figure 21.

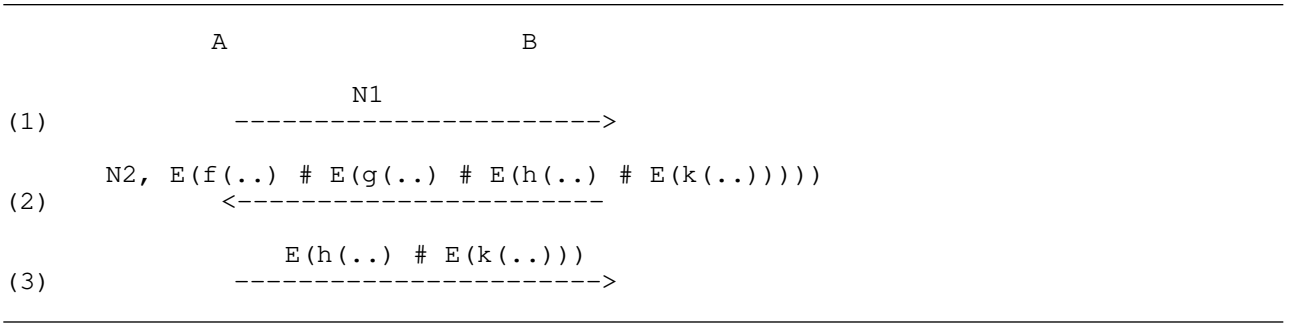

Figure 22. Canonical protocol 8: Cryptographic Hardware support

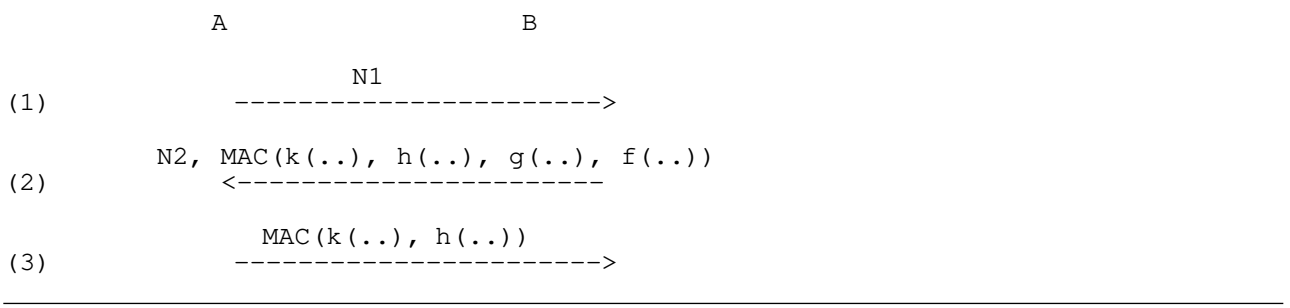

Figure 23. Canonical protocol 9: Exportability with hardware support

Each of the required three MAC computations is a subset of the previous one, so that an efficient implementation should compute them starting from MAC(h(..)) (i.e. the last one) and save intermediate results for later re-use. If a hardware implementation of a MAC algorithm is available, the cost of a driver invocation for that hardware may actually be more substantial than the cost of an extra cryptographic block operation. In that case, a canonical protocol using just two MAC operations is preferable. However a fourth block encryption is then desirable to make known-plaintext attacks of flow (2) harder. This is represented in Figure 22. In a DES context with all # operators designating XOR operations, this amounts to the MAC-based protocol represented in Figure 23. Again an efficient implementation could compute the second MAC expression as part of the first one and save it for reuse in flow (3).

The export of protocols based on canonical forms 7 or 9 should pose no problem if the DES function hardware or software (object code only) is restricted to MAC operations and does not provide any access to full-function encryption and decryption, i.e. if the DES is provided strictly as a one-way hash function but otherwise unusable as an encryption-decryption tool. Export of the source code of a software implementation would however remain subject to controls because the source could clearly be exploited to achieve full-function encryption and decryption.

All export issues can however be waived if a true, plain, one-way hash function is used for MAC instead of DES CBC. For instance, the SHA function Smid92“ recently proposed by NIST, or the MD4 or MD5 hash functions Rivest92a, Rivest92b“ can be used with secret prefixes or suffixes for message authentication Tsudik92“. In this case, the size of the MAC expressions could be 128 instead of 64 bits, although they could be truncated at or folded onto 64 bits (which would of course weaken the strength of the original MD4 and MD5 MAC but may still be sufficient in many scenarios). This is the approach taken in the KryptoKnight system Molva92“.

3.4. Specific Examples

Example With Two Encryptions

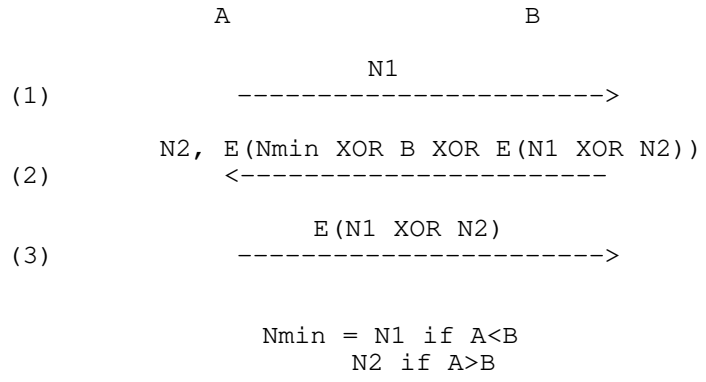

Figure 24. Secure protocol with two encryptions

Figure 24 shows an example of a protocol secure against interleaving attacks with only two encryptions, where f() is the exclusive-or of the name of the responder with the nonce of the party with the lower address (or name), while g() is the exclusive-or of the two nonces. The name B appearing in flow (2) is assumed to be no longer than nonces and is used here as the direction indicator D. Since this protocol uses only two encryptions, it is not immune to known-plaintext attacks.

Example With Three Encryptions

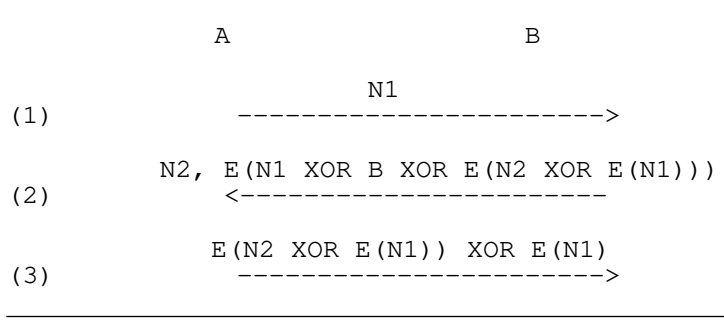

Figure 25. Secure protocol with three encryptions

By adding an extra block encryption operation inside g(), and re-using its result by exclusive-oring it with the message of flow (3), one gets a protocol that makes known-plaintext attacks harder. This is depicted in Figure 25 That protocol is otherwise also secure against interleaving attacks because its function g() is the exclusive-or of N2 with E(N1), and since E(N1) does not appear anywhere in cleartext, an intruder can never obtain or derive g() from past observations. A protocol very close to this one was even proven secure in a broader sense in Bird91".

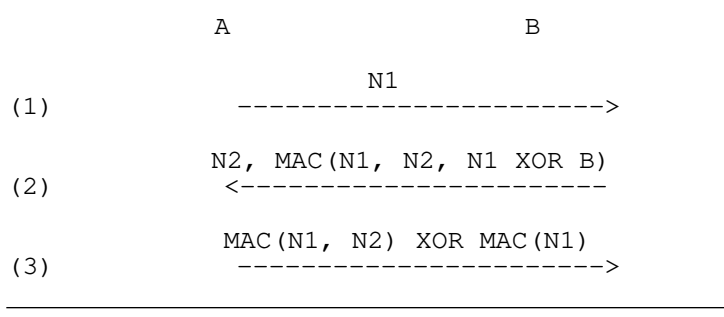

Figure 26. Secure protocol with three MAC operations and three encryptions.

The above protocol can of course be expressed in terms of MAC operations rather than encryption operations to underline its exportability, as represented in Figure 26.

Example With Two MAC Operations and Four Encryptions

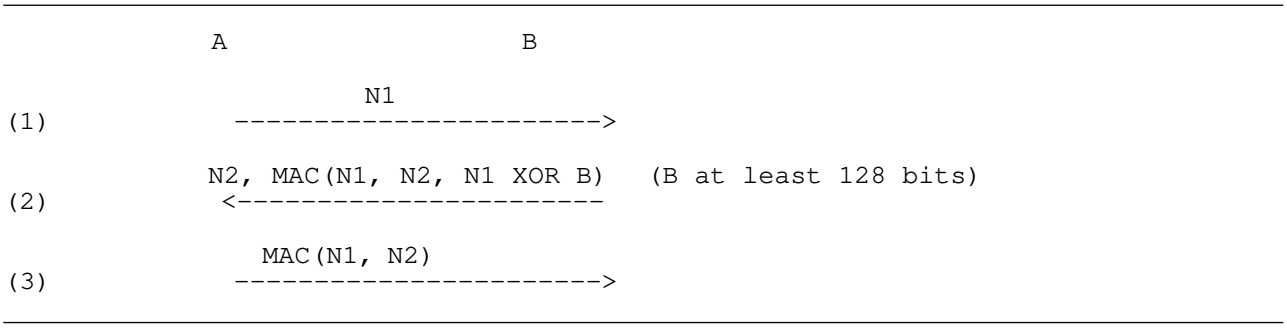

Figure 27. Secure protocol with two MAC operations and four encryptions

By a similar minor extension, one can easily produce a protocol suited for environments where hardware support for MAC computations is available and the cost of invoking that hardware is more important than the cost of an individual block operation: this protocol uses at least six encryption operations (only four if optimized to re-use two) but only two invocations to the MAC hardware, as depicted in Figure 27 Four of the encryption operations come from the operations on N1 and N2 while the remaining two (or more) come from operations on the expression (N1 XOR B). The XOR in this expression denotes an operation whereby the nonce N1 is XOR'ed repeatedly with each successive cleartext block in the name B to add randomness to that name between different executions of the protocol. The name B must be padded to at least two cryptographic blocks to protect flow (2) against direct known-plaintext attacks that would otherwise be possible. The protocol is then as secure as the previous one and for the same reason that g() cannot be obtained by any intruder.

4. Conclusion

We first described a few examples of interleaving attacks on simple two-way authentication protocols to derive a set of security criteria that protocols should meet to resist such attacks. Based on these examples, we suggested that the design of such protocols is not a trivial exercise. Thus we set out to derive intuitively as simple as possible a canonical form for two-way authentication protocols that were tested systematically against interleaving attacks, yet are simple enough to be usable within the lower layers of a networking architecture. We gave several practical and elegant examples of such realizable protocols.

From the last of these examples, one might sweepingly conclude that plain MAC operations on sufficiently large numbers of authentication parameters such as nonces and names of involved parties meet all requirements. Admittedly, this conclusion is no real surprise and may sound intuitive. The interest of the design, the canonical protocol and the systematic test of security we have developed is however three-fold:

1. Experience has shown that while using MAC operations should work intuitively, the choice and order of the arguments to these operations cannot be arbitrary. There are certain requirements for asymmetry and diversity. Many protocols in use today are so complex that they probably meet these asymmetry and diversity requirements, but at the cost of many unnecessary and expensive cryptographic operations. On the other hand, several of today's simplest protocols are actually too simple, do not meet the asymmetry and diversity requirements and thus do not resist certain attacks. Thus we have put a lower bound on the number and the form of terms that a MAC-based authentication protocol must contain to be secure.

2. The canonical form we have developed covers far more than just DES-based MAC protocols. It clearly suggests that there is a wealth of other possible functions and protocols to achieve secure two-way authentication. Specifically, there is no requirement to use any cryptographic function. Any good one-way hash function would do Gong89“. For instance, one could use an SHA function Smid92“, or an MD4 or MD5 function

Rivest92a, Rivest92b“ suitably seeded with a secret prefix to implement the MAC functions of the canonical form Tsudik92“.

3. Finally, the kind of security test that we provided on the canonical protocol has never been performed before on any existing protocol we are aware of.

The canonical two-way authentication protocol described in this paper actually forms the basis for a family of modular and parametrized security protocols offering such functions as key distribution, single network sign-on, password management, and related security services in a distributed environment. The entire family benefits from the test of security provided for the basic authentication protocol. All protocols in the family allow optional use of time-stamps instead of nonces, require minimal overhead and minimal-size cryptographic tokens, are amenable to use for securing network mechanisms at any architectural layer, and rely strictly on exportable data integrity mechanisms such as one-way hash functions rather than on confidentiality mechanisms. A prototype implementation of the protocol family using MD5 technology has been implemented on AIX, IBM's Unix product Molva92“. It operates both in TCP/IP and SNA environments. Further details on the entire protocol family will be documented in a later publication.

Acknowledgements

We would like to thank Reid Sayre for initially alerting us to the problem and the difficulties of designing simple yet secure authentication protocols for low-level networking mechanisms. We thank Liba Svobodova for her numerous comments on successive versions of this paper. We also thank the JSAC reviewers for their help in properly positioning and clarifying certain statements, especially assumptions and complex arguments in the section about the security testing methodology.

Bibliography

Bauer83 R. K. Bauer, T. A. Berson, R.J. Freiertag, "A key distribution protocol using event markers," ACM TOCS 1 3 (1983) 249-255.

Bellovin90 S. M. Bellovin, M. Merritt, "Limitations of the Kerberos authentication system", ACM CCR 20 5 (Oct.90) 119-132.

Biham90 E. Biham, A. Shamir, "Differential Cryptanalysis of DES-like Crypto-systems", Proc. Crypto'90 (Jun.90).

Bird91 R. Bird, et al., "Systematic design of two-party authentication protocols," Proc. Crypto 91, Santa Barbara, CA (Aug.91) 44-61, available as "Advances in Cryptology", J. Feigenbaum (Ed.), Lecture Notes in Comp. Sc. 576, Springer Verlag (1991).

Burrows89 M. Burrows, M. Abadi, R. M. Needham, "A logic of authentication", Proc. 12th ACM SOSP, ACM OSR 23 5 (Dec.89) 1-13.

Denning81 D. E. Denning, G. M. Sacco, "Timestamps in key distribution systems", CACM 24 8 (Aug.81) 533-536.

DES77 "Data Encryption Standard", FIPS 46, NBS (Jan.77).

Gong89 L. Gong, "Using one-way functions for authentication", ACM CCR 19 5 (Oct.89) 8-11.

Ianson90 C. I'Anson, C. Mitchell, "Security defects in CCITT Recommendation X.509 - The Directory Authentication Framework", ACM CR 20 2 (Apr.90) 30-34.

ISOSC27 "Entity Authentication Using Symmetric Techniques", ISO-IEC JTC1.27.02.2(20.03.1.2) (Jun.90).

ISO8732 "Banking - Key management (wholesale)" ISO 8732, Geneva (1988).

ISO9594 "OSI Directory - Part 8: Authentication Framework", ISO 9594-8, Geneva (1988).

Jueneman85 J. J. Jueneman, S. M. Matyas, and C. H. Meyer, "Message Authentication", IEEE Communication Magazine, (1985) 29-40.

Meyer82 C. H. Meyer and S. M. Matyas, "Cryptography: a new dimension in computer data security", Wiley, New York (1982).

Molva92 R. Molva, G. Tsudik, E. Van Herreweghen, S. Zatti, "KryptoKnight authentication and key distribution system," submitted to ESORICS 92, Toulouse, France (23-25 Nov.92).

Morris79 R.Morris, K.Thompson, "Password security: a case history," CACM 22 11 (1979) 594-597.

Needham78 R. M. Needham, M. D. Schroeder, "Using encryption for authentication in large networks of computers," CACM 21 12 (1978) 993-998.

Otway87 D. Otway, O. Rees, "Efficient and timely mutual authentication", ACM OSR 21 1 (Jan.87) 8-10.

RSA83 R. L. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key crypto-systems," CACM 21 2 (1978) 120-126 & CACM 26 1 (1983) 96-99.

Rivest92a R. Rivest, "The MD4 Message Digest Algorithm", Internet RFC 1320 (Apr.92).

Rivest92b R. Rivest, "The MD5 Message Digest Algorithm", Internet RFS 1321 (Apr.92).

Smid92 M. E. Smid, D. K. Branstad, "Response to the comments on the NIST proposed digital signature standard', to appear in Proc. Crypto 92.

Spafford89 E. H. Spafford, "The Internet worm program", ACM CCR 19 1 (Jan.89) 17-57.

Steiner88 J. G. Steiner, et al., "Kerberos: an authentication server for open network systems", Proc. Usenix Conf. (Winter 88).

Stubblebine92 S. G. Stubblebine, V. D. Gligor, "On message integrity in cryptographic protocols", to appear in Proc. IEEE Symp. on Res. in Sec. & Priv., Oakland, CA (May 92).

Tsudik92 G. Tsudik, "Message authentication with one-way hash functions," Proc. IEEE Infocom 92, Florence, Italy (May 92)

Biographical Notes

Ray Bird is a Senior Engineer in the Networking Systems Architecture group at IBM, Research Triangle Park, North Carolina. He joined IBM in 1967 after obtaining a BS in mathematics from Rutgers Unversity. From 1967 to 1977 he worked on the OS/360 operating system and the Telecommunications Access Method (TCAM). From 1977 to the present he has worked on Systems Network Architecture protocols including subarea, LU 6.2, and Advanced Peer-to-Peer Networking (APPN). He is currently working on future enhancements to APPN.

Inder Gopal's biography and picture recently appeared in JSAC Vol. 19, No. 7, p 966.

Amir Herzberg received the B.Sc. in Computer Engineering in 1982, the M.Sc. in Electrical Engineering in 1986, and the D.Sc. in Computer Science in 1991, all from the Technion, Israel Institute of Technology. He is interested in security and in protocols for communication networks, and in particular in cryptographic protocols. He is also interested in robust network protocols, software protection, computer viruses, realistic analysis of distributed algorithms, and consumer applications of high speed networks.

Philippe A. Janson received the B.S. in Electrical Engineering from the University of Brussels, Belgium in 1972, and the M.S., E.E., and Ph.D. in Computer Science from the Massachusetts Institute of Technology, USA, in 1974, 1975, and 1976, respectively.

From 1974 to 1976, he was a Research Assistant at the M.I.T. Laboratory for Computer Science, working on the internal structure of Multics. Since 1977, he is with the IBM Zurich Research Laboratory in Switzerland, where he has been working on high-speed packet switches, the IBM Token Ring, LAN servers and gateways. During 1986-87, he was on assignment with the IBM Development Laboratory in Austin, Texas, working on LAN gateways for OS/2. Since 1987 he has been managing various projects dealing with the integration of heterogeneous networks, and more recently network security. He is a Visiting Professor at the Department of Engineering of the Free University of Brussels where he has been teaching an Operating Systems course since 1976. He has also taught various courses on operating systems, LAN's, OSI, and network security at the IBM International Education Center and at various professional organizations and international conferences.

His research interests include operating systems, distributed systems, and computer communication. He is the author of several papers and patents in these fields, as well as a textbook on operating systems. He is a member of the IEEE Computer Society, the ACM, SIGOPS and SIGCOMM, and Sigma Xi. He received a Harkness Fellowship in 1972 and several IBM invention and technical awards since then.

Shay Kutten has received his B.Sc in Computer Science (cum-laude) from the Technion, Israel in 1981, and his M.Sc (cum-laude) from the Technion in 1984, on the subject of "Broadcasting in Radio Networks". He received his D.Sc in Computer Science from the Technion in 1988, on the subject of Distributed Traversal and Leader Election. Until 1989 he was a post doctoral fellow in the communication department in IBM T.J. Watson Research Center, where he is now a Research Staff Member, and the project leader of the Network Architecture and Algorithms group. This group designs distributed protocols for control of high-speed networks, network security protocols and does research in others areas of distributed algorithms and cryptography.

Refik Molva obtained a Ph.D. in Computer Science from the Paul Sabatier University in Toulouse, France, in 1986. He then joined the IBM Zurich Research Laboratory where he worked on network interconnection and network security projects. He obtained an IBM Outsanding Technical Achievement award and he is the co-author of several patents on network security mechanisms. Since March 1992 he is an assistant professor at the Corporate Communications Department of the Eurecom Institute in France. His current research interests include security in intelligent and mobile networks and service creation environments for telecommunication networks.

Moti Yung received a Ph.D. in computer science from Columbia University in 1988, and B.Sc. and and M.Sc. in mathematics from Tel-Aviv University in 1979, and 1983 respectively. During 1988 he was a visiting scientist at IBM Almaden Research Center, and since 1989 he is with IBM T.J. Watson Research Center. His research interests and publications include the areas of cryptography, network security, design and analysis of algorithms, theory of computations, distributed and parallel computing, and communication architectures.