Authentication, Key Distribution, and Secure Broadcast in Computer Networks Using No Encryption or Decryption

Li Gong

SRI International
Computer Science Laboratory
333 Ravenswood Avenue
Menlo Park, California 94025 U.S.A.
gong@csl.sri.com

May 23, 1994

Abstract. This paper describes Needham-Schroeder type authentication and key-distribution protocols that do not use encryption or decryption. The new protocols use polynomial interpolation instead, and one-way hash functions if necessary. These protocols are simple, elegant, and efficient when compared with conventional protocols based on encryption. One significant advantage of this new approach is that critical security properties of the protocols no longer depend on an underlying cryptosystem being secure. Protocols for secure broadcast and for secure communication are also developed using the same techniques.

1 Authentication

Authentication, the process of identifying entities such as users and machines, is a crucial element in secure distributed systems and networks. The process of authentication is often coupled with key distribution, and some argue that the two issues should be considered jointly rather than separately [3, 14].

Research in authentication protocols has been significantly influenced by the work of Needham and Schroeder [14], and thus we confine this paper to the discussion of Needham-Schroeder type authentication protocols. Typically, for two clients to authenticate each other, an authentication server distributes a session key to both clients so that they will have a secure communication channel. The protocols can be based on symmetric-key cryptosystems or asymmetric-key (e.g., public-key) cryptosystems [5, 14]. The server either shares a secret key with each client, or the client (securely) registers a public key with the server in advance [14]. In the rest of our discussion, we examine only the case in which server and clients share symmetric keys.

In an effort to reduce the use of encryption in authentication protocols (partly to increase protocol efficiency), Gong describes an authentication protocol that uses only secure keyed one-way hash functions[1] [7]. Basically, for the server (call it S) to distribute a session key k to clients A and B, S chooses k such that it is computable in a one-way hash fashion from the secret key that A and S share, so that, instead of sending the key k to A, the server needs to send only additional information for computing the one-way hash function. Because the hash function is one-way, the same process cannot be repeated with B. But S can send k to B using a simple exclusive-or operation to hide k. This technique was subsequently used elsewhere to design efficient authentication protocols for computer networks (e.g., [2]).

In this paper, we describe new authentication protocols that do not use encryption or decryption. These protocols are symmetric in the sense that all clients use the same routine to compute the session key. The protocols are novel in that they are based on polynomial interpolation [11, pp.484-486]. Very significantly, the new protocols (unlike existing ones) do not depend on the underlying cryptosystem being secure. Moreover, the new protocols do not use techniques that are subject to licensing or export controls.

The new techniques can also be used for developing secure broadcast protocols and for secure communication. We discuss these topics in Sections 4 and 5.

2 Basic Techniques

In this section, we start by describing the basic technique for ensuring the secrecy of the distributed session key. We then describe improvements that protect against attacks by insiders, attacks on integrity, and replay attacks. For some applications, not all these improvements are necessary.

Due to the many ways in which these techniques can be combined, our full protocols in later sections use only some of the techniques described here, although they are all applicable to authentication, key distribution, and secure broadcast.

2.1 Secrecy

Suppose the server S shares a secret k_1 with client A and another secret k_2 with client B.[2] To distribute a session key to the clients, S first selects a session key k, and then solves the following equations for a_1 and a_2:

$$\begin{cases} k + a_1 \times 1 + a_2 \times (1)^2 &= k_1 \\ k + a_1 \times 2 + a_2 \times (2)^2 &= k_2 \end{cases}$$

All computations are performed $mod(q)$ where q is a suitably large prime number [11]. Let $f(x) = k + a_1 \times x + a_2 \times (x)^2$. Server S now calculates $f(3)$ and $f(4)$, and sends them to both A and B.

[1] See [1] for a formulation of the concept and a proof of the existence of secure keyed one-way hash functions.

[2] If $k_1 = k_2$, then $a_1 = a_2 = 0$ and $k = k_1$. Later we discuss how to avoid such situations.

The above procedure is like the "reverse calculation" of secret sharing in that, instead of calculating some arbitrary "shadows" of a given secret [16], server S is given some pieces of "shadows" that have been predetermined (e.g., k_1 and k_2), and is then required to calculate what the rest of the "shadows" should be.

A knows k_1, and thus can use interpolation to solve the following equations to obtain k [11]:

$$\begin{cases} k + a_1 \times 1 + a_2 \times (1)^2 &= k_1 \\ k + a_1 \times 3 + a_2 \times (3)^2 &= f(3) \\ k + a_1 \times 4 + a_2 \times (4)^2 &= f(4) \end{cases}$$

Similarly, B knows k_2, and thus can obtain k by solving the following equations:

$$\begin{cases} k + a_1 \times 2 + a_2 \times (2)^2 &= k_2 \\ k + a_1 \times 3 + a_2 \times (3)^2 &= f(3) \\ k + a_1 \times 4 + a_2 \times (4)^2 &= f(4) \end{cases}$$

An attacker who monitors network traffic can gain absolutely no information of k, because for any value of k, there is a pair of a_1 and a_2 (and therefore a pair of k_1 and k_2) that will satisfy the above equations. This argument is identical to that used in Shamir's secret-sharing scheme [16], and the guarantee of k's secrecy is information theoretic. By the same argument, the attacker cannot gain any information about k_1 or k_2.

2.2 Prevention of Insider Attacks

In the preliminary protocol described in the previous section, an insider can attack more effectively. For example, after the authentication, A knows k, a_1, and a_2, and thus can easily compute k_2, which is $f(2)$. An outsider can attack in the same way if k is later compromised through other means.

One solution is to use a one-way hash function $h()$, such as MD-5 [15] or Snefru [12], but in a keyed fashion [1]. The server selects a random number r and solves the following equations instead:

$$\begin{cases} k + a_1 \times 1 + a_2 \times (1)^2 &= h(k_1, r) \\ k + a_1 \times 2 + a_2 \times (2)^2 &= h(k_2, r) \end{cases}$$

The server then sends $f(3)$, $f(4)$, and r to the clients. Here, an outside attacker still cannot compute k because he does not know either k_1 or k_2 and thus cannot compute $h(k_1, r)$ or $h(k_2, r)$, given that $h()$ is a secure keyed one-way hash function.

Moreover, A can compute only $h(k_2, r)$, which is used as a one-time pad and cannot lead to the compromise of k_2. Note that now the secrecy argument against insider attacks is no longer information theoretic. This is because A can compute $h(k_2, r)$, and thus can guess a value of k_2 (call it k_2') and verify the guess by computing $h(k_2', r)$ and then comparing it with $h(k_2, r)$. A

match indicates that $k_2 = k_2'$ with a high probability. However, if we assume that the selection of k_2 is from a sufficiently large space and by random means, then such an exhaustive attack is computationally infeasible.

The number r should not be reused. Suppose r is first used to distribute a session to A and B, and is later reused to distribute a session key to B and C. Now since A is able to compute $h(k_2, r)$ during the first protocol execution, A can obtain the new session key between B and C.

We observe that if A can intercept the server's message to B and replace it with some other message, then, because A knows $h(k_2, r)$, A can in theory sends a different key k' to B by modifying $f(3)$ and $f(4)$. This type of attack is often possible when the message delivery system is unreliable or slow, especially when broadcast is simulated in software instead of implemented in hardware [9]. But since k is shared only between A and B, it is not unclear what advantages A gains by cheating this way except that A can make B use a key favored by A. However, when we develop broadcast protocols later in this paper, we see that it is desirable to ensure that A (or any other client) cannot change k in any predictable way.

Another potential security problem is that, if A and B happen to share the same secret with the server, i.e., $k_1 = k_2$, then they will be able to discover this fact during authentication. As we already argued, A can compute $h(k_2, r)$. If this value matches $h(k_1, r)$, then A will know that $k_1 = k_2$ with a very high probability. Future sessions of authentication will be able to confirm whether $k_1 = k_2$. One solution to this last problem is not to use the same nonce r, e.g., by using $h(k_1, r)$ and $h(k_2, r+1)$ instead. Alternatively, we can include the identifications A and B in the computation, e.g., by using $h(k_1, A, r)$, and $h(k_2, B, r)$.

A better protocol that solves all the above problems is to let the server solve the following equations:

$$\begin{cases} k + a_1 \times h(k_1, r) + a_2 \times (h(k_1, r))^2 &= k_1 \\ k + a_1 \times h(k_2, r) + a_2 \times (h(k_2, r))^2 &= k_2 \end{cases}$$

Now, A cannot use $f()$ to attack B because A does not know $h(k_2, r)$, the value for which B computes $f()$. In particular, A, by modifying $f(3)$ and $f(4)$, cannot modify k in any predictable fashion.

Because $h()$ has collisions, in case no solution can be found for the above equations, S can select another value for r and repeat the process. Alternatively, we can attach one more bit to the output of hash functions, so that $h(k_1, r) \mid 0 \neq h(k_2, r) \mid 1$, where "$x \mid y$" denotes x concatenated with y. This usage has the benefit that even if k_1 happens to be equal to k_2, the hash values cannot be identical.

It seems a good practice not to use shared keys such as k_1 directly, so we can let the server solve the following equations:

$$\begin{cases} k + a_1 \times h(k_1, r) + a_2 \times (h(k_1, r))^2 &= h(k_1, r+1) \\ k + a_1 \times h(k_2, r) + a_2 \times (h(k_2, r))^2 &= h(k_2, r+1) \end{cases}$$

The reason for using $h(k_1, r+1)$ instead of $h(k_1, r)$ is again to prevent B from solving a second-order equation to compute $h(k_1, r)$.

Finally, we speculate that the use of one-way (hash) function (or another similar mechanism) cannot be avoided, because otherwise for A to find out B's key k_2, it is just a matter of solving a higher order equation.

2.3 Integrity

In the previous section, we discussed how to prevent an (inside) attacker from modifying k in any predictable way. For many applications, this type of authentication is sufficient because as soon as the session key is used in subsequent handshake or communication, a client will notice that the key he received has been modified.

In other applications, it is desirable or even vital for a client to know that the session key he receives is genuine, before he uses it. To ensure key integrity, there are at least two solutions.

One solution is to add redundancy to the key that is sent, so that any modification en route will be detected. For example, typically k is 64 bit long and the hash function $h()$ has an output size of 128 bits. The server can distribute $k \mid k$ instead of k. In other words, S solves the following equations:

$$\begin{cases} (k \mid k) + a_1 \times h(k_1, r) + a_2 \times (h(k_1, r))^2 &= h(k_1, r+1) \\ (k \mid k) + a_1 \times h(k_2, r) + a_2 \times (h(k_2, r))^2 &= h(k_2, r+1) \end{cases}$$

In this protocol, A (or B) solves the equations as before, checks that in the result the first half is identical to the second half, and then accepts the first half as the session key. Any modification to the server's message highly likely will cause A (or B) to retrieve a value that is not of the form $k \mid k$.

Another solution is to use a one-way hash function to introduce redundancy. For example, the server can send A an additional data item, $h(k_1, k, r, A, B)$, as a secure checksum. After retrieving the session key, A can recompute this checksum and compare it with the received value. In order to forge a seemingly legitimate checksum, the attacker needs to know k_1.

2.4 Freshness

In protocols described in previous sections, an attacker may record the server's message from an earlier run of the protocol and replay it later. If the session key k has been compromised between the two runs, a successful replay attack will lead to a security breach.

To detect replay attacks, the server must include a freshness identifier in his message, so that each client can be convinced that the message is indeed fresh. The freshness identifier must be securely bound to the message in such a way that an attacker cannot modify the identifier without being detected.

A freshness identifier can be the server's timestamp, if the server and the clients have securely and closely synchronized clocks. It can also be a nonce, such as a random number, generated by the intended recipient. Let f_{id} denote a freshness identifier. Then one possible protocol is to let the server solve the following equations:

$$\begin{cases} (k \mid k) + a_1 \times h(k_1, f_{id1}, r) + a_2 \times (h(k_1, f_{id1}, r))^2 &= h(k_1, f_{id1}, r+1) \\ (k \mid k) + a_1 \times h(k_2, f_{id2}, r) + a_2 \times (h(k_2, f_{id2}, r))^2 &= h(k_2, f_{id2}, r+1) \end{cases}$$

Here, f_{id1} and f_{id2} can be identical (when they are the server's timestamp) or different (when they are the clients' respective nonces). There are many other ways in which the freshness identifiers can be embedded. For example, the server can send A a secure checksum, $h(k_1, f_{id1}, k, r, A, B)$ that is also fresh. A timestamp needs to be sent as cleartext in the server's message, unless the recipients can accurately predict its value.

3 New Authentication Protocols

Based on the basic techniques we have introduced so far, we now give concrete authentication protocols. We first discuss the case where the authentication server distributes a key to two clients. To avoid repetition, we give only a nonce-based protocol. In Section 3.2, we generalize the protocol to the n-client case, for which we give a timestamp-based protocol. Other variations can be easily derived. We call these protocols NED-A(n), for authentication with No Encryption or Decryption, where n is the number of clients.

3.1 Protocol NED-A(2) Using Nonces

Suppose the server S shares a secret k_1 with client A, and a secret k_2 with client B. Let I_a denote A's identification number, such as its Internet domain name or IP address (possibly hashed), and similarly I_b for B. For ease of discussion, these identification numbers are assumed to be distinct values and neither of them are 1 or 2. Let N_a, N_b, and r denote the nonce chosen by A, B, and S respectively. We use $S \rightarrow A : x, y$ to denote that S sends A a message that is the concatenation of x and y. The NED-A(2) protocol without handshake is as follows.

1. A $\rightarrow$ B: I_a, I_b, N_a
2. B $\rightarrow$ S: I_a, I_b, N_a, N_b
3. S $\rightarrow$ B: $r, f(1), f(2), h(k_2, N_b, k, r, I_b, I_a)$
4. S $\rightarrow$ A: $r, f(1), f(2), h(k_1, N_a, k, r, I_a, I_b)$

Informally, the protocol works in the following way. In message 1, A tells B that an authentication protocol is initiated and sends along A's nonce. In message 2, B sends both clients' identifiers and nonces to the server. Server S then selects a key k and a nonce r, and solves the following equations for a_1 and a_2:

$$\begin{cases} k + a_1 \times h(k_1, I_a, r) + a_2 \times (h(k_1, I_a, r))^2 &= h(k_1, I_a, r+1) \\ k + a_1 \times h(k_2, I_b, r) + a_2 \times (h(k_2, I_b, r))^2 &= h(k_2, I_b, r+1) \end{cases}$$

The reason for including I_a and I_b in formulae $h(k_1, I_a, r)$ and $h(k_2, I_a, r)$ in the equations is to ensure that even if A and B happen to share the same secret with the server, i.e., $k_1 = k_2$, the equations are likely to have a solution.

Let $f(x) = k + a_1 \times x + a_2 \times (x)^2$. Server S now calculates $f(1)$ and $f(2)$, computes the hash functions, and sends messages 3 and 4. A then uses interpolation to solve the following equations to obtain k.

$$\begin{cases} k + a_1 \times h(k_1, I_a, r) + a_2 \times (h(k_1, I_a, r))^2 &= h(k_1, I_a, r+1) \\ k + a_1 \times 1 + a_2 \times (1)^2 &= f(1) \\ k + a_1 \times 2 + a_2 \times (2)^2 &= f(2) \end{cases}$$

A then uses this k to recompute $h(k_1, N_a, k, r, I_a, I_b)$ and compares it with the received value. A match indicates that the message originates from the server S and has not been modified during transmission, and therefore k is indeed the session key the server has distributed. Upon finding a mismatch, A terminates the protocol run or takes appropriate actions. B's action is the same as A.

If A and B want to complete a two-way handshake, two optional messages can be added to the protocol:

5. B $\rightarrow$ A: $h(k, N_a, I_a, I_b), N_b$
6. A $\rightarrow$ B: $h(k, N_b, I_b, I_a)$

Note that B obtains A's nonce in message 2. Also, if timestamps are used instead of nonces, the order of I_a and I_b in the calculation of $h()$ in the two handshake messages must be different because otherwise, the two hash values could be identical (namely when A's and B's timestamps are identical), in which case an attacker can playback B's own message 5 in place of message 6.

We can of course combine authentication and handshake to obtain a more compact protocol with only five messages:

1. A $\rightarrow$ B: I_a, I_b, N_a
2. B $\rightarrow$ S: I_a, I_b, N_a, N_b
3. S $\rightarrow$ B: $r, f(1), f(2), h(k_1, N_a, k, r, I_a, I_b), h(k_2, N_b, k, r, I_b, I_a)$
4. B $\rightarrow$ A: $r, f(1), f(2), h(k_1, N_a, k, r, I_a, I_b), h(k, N_a, I_a, I_b), N_b$
5. A $\rightarrow$ B: $h(k, N_b, I_b, I_a)$

The security arguments for this protocol are the same as those given in Section 2. In addition, in an exhaustive cryptanalysis, the attacker may attempt to build up a dictionary of possible session

keys and hash values [6]. One way to increase the difficulty of this attack is to use longer texts in the input to the hash function. This method is likely to greatly increase the search space with little lost of efficiency, because the computation of a hash function is extremely fast and the length of the hash value remains constant.

3.2 Protocol NED-A(n) Using Timestamps

We now describe a protocol for the n-client case. Suppose server S is to distribute a secret to n clients P_i, $i = 1, 2, \ldots, n$, where I_i is P_i's identification number, and $I_i \neq I_j$ when $i \neq j$. For simplicity of discussion, we assume that these numbers are outside the range of 1 to n. Also suppose that S and P_i share a secret k_i. All computations are performed $mod(q)$ where q is a suitably large prime number. The protocol NED-A(n) is as follows.

The server selects a secret k and a random number r. It then solves the following equations for a_i, $i = 1, 2, \ldots, n$.

$$\begin{cases} k + a_1 \times h(k_1, I_1, r) + \ldots + a_n \times (h(k_1, I_1, r))^n &= h(k_1, I_1, r+1) \\ k + a_1 \times h(k_2, I_2, r) + \ldots + a_n \times (h(k_2, I_2, r))^n &= h(k_2, I_2, r+1) \\ \qquad\qquad \ldots & \qquad\qquad \ldots \\ k + a_1 \times h(k_n, I_n, r) + \ldots + a_n \times (h(k_n, I_n, r))^n &= h(k_n, I_n, r+1) \end{cases}$$

Let $f(x) = k + a_1 \times x + \ldots + a_n \times (x)^n$. The server now calculates $f(i)$, $i = 1, 2, \ldots, n$, and sends the following messages, where T_s is S's timestamp:

$$i. \quad \mathrm{S} \rightarrow P_i: \quad I_1, I_2, \ldots, I_n, r, T_s, f(1), f(2), \ldots, f(n), h(k_i, I_1, I_2, \ldots, I_n, T_s, r, k)$$

Similar to NED-A(2), P_i can use interpolation to retrieve k from the following equations:

$$\begin{cases} k + a_1 \times 1 + \ldots + a_n \times (1)^n &= f(1) \\ k + a_1 \times 2 + \ldots + a_n \times (2)^n &= f(2) \\ \qquad\qquad \ldots & \qquad \ldots \\ k + a_1 \times n + \ldots + a_n \times (n)^n &= f(n) \\ k + a_1 \times h(k_i, I_i, r) + \ldots + a_n \times (h(k_i, I_i, r))^n &= h(k_i, I_i, r+1) \end{cases}$$

P_i then recomputes $h(k_i, I_1, I_2, \ldots, I_n, T_s, r, k)$ to check that the message comes from S and has not been modified during transmission. Handshake messages can be added if required. The security arguments are identical to that for NED-A(2).

The eliminate the need for sending the n additional hash values, S can solve

$$\begin{cases} (k \mid k) + a_1 \times h(k_1, I_1, r) + \ldots + a_n \times (h(k_1, I_1, r))^n &= h(k_1, I_1, I_2, \ldots, I_n, T_s, r, 1) \\ (k \mid k) + a_1 \times h(k_2, I_2, r) + \ldots + a_n \times (h(k_2, I_2, r))^n &= h(k_2, I_1, I_2, \ldots, I_n, T_s, r, 2) \\ \qquad\qquad \ldots & \qquad\qquad \ldots \\ (k \mid k) + a_1 \times h(k_n, I_n, r) + \ldots + a_n \times (h(k_n, I_n, r))^n &= h(k_n, I_1, I_2, \ldots, I_n, T_s, r, n) \end{cases}$$

Here, the identities of the n clients are bound by the hash values $h(k_i, I_1, I_2, \ldots, I_n, T_s, r, i)$, which also have freshness identifiers embedded. The use of $k \mid k$ provides redundancy for integrity check. In this protocol, the server needs to send only the following messages, which can be sent in a single broadcast:

$$i. \quad \mathrm{S} \rightarrow P_i\text{: } I_1, I_2, \ldots, I_n, r, T_s, f(1), f(2), \ldots, f(n)$$

The same technique can also be used to reduce message length in NED-A(2). Note that if nonces are used and the server uses a single broadcast, then the amount of reduction may be decreased because a client may need to know the nonces of other clients.

3.3 Advantages of the New Approach

The NED-A protocols offer a number of significant advantages:

- **Fewer assumptions about the cryptosystems**. Most of previously published protocols must depend on the assumption that the underlying cryptosystems are secure against cryptanalysis. In our approach, the privacy of the distributed session key is protected because a potential attacker cannot obtain any information about the session key, except by exhaustive search. Here, the hash function must be one way, but it is not a serious problem if a small number of collisions are known, because we can add more texts to the hash-function input in order to avoid the collisions.
- **Efficiency**. To distribute a session key to n clients, the server and the clients each computes $2n$ hash functions, and solves n equations with n unknowns, where the computational complexity for polynomial interpolation is only $O(n(\log n)^2)$ [11]. In existing protocols, the server does n encryptions and each client does one decryption. Since it is widely accepted that a one-way hash function is much cheaper to compute than a traditional encryption algorithm such as DES [17], the NED-A(n) protocols are more efficient, especially when n is small. Note that the size of q (in $mod(q)$) needs to increase only logarithmically when n increases.
- **Free**. Since the protocols use polynomial interpolation and hash functions, which are in the public domain, the protocols, together with related algorithms for secure broadcast and secure communication described in Sections 4 and 5, are free of licensing and export controls.
- **Fewer assumptions about the one-way hash functions**. In previous works where the use of a cryptosystem is replaced by the use of a one-way hash function, a bit-wise exclusive-or operation is generally used to protect data secrecy [2, 7]. This usage implicitly requires additional properties about the statistical distribution of the output from the one-way hash function. These properties probably hold for well-designed hash functions, but they are not required in the theoretical definition of a one-way hash function [12, 15]. For example, given that $h()$ is a one-way hash function, we define $g(x) = h(x) \mid 1 \cdots 1$, i.e., the output of $h()$ concatenated with a string of 1's. We can easily verify that $g()$ is also a one-way hash function, but clearly it is not suitable for bit-wise exclusive-or operation. Our protocols do not require these additional properties.

Other improvements are possible. For example, if we can assume that, in some environment, secrets shared with the server are all distinct, then we can simplify some computations. We do not further discuss the many possible variations of the protocols we present in this paper, which are mainly for demonstrating the new principles and techniques.

4 Secure Broadcast

If the server can distribute a session key to n clients, the server can also broadcast a (secret) message to the n clients. For example, we can use the first NED-A(n) protocol and replace the session key k with a message m. In such a broadcast, unlike in authentication, a client may not need to know exactly all the identities of other clients. When m is long, it can be broken into smaller blocks. We call such protocols NED-B, for Broadcast with No Encryption or Decryption.

The server solves the following equations to broadcast m:

$$\begin{cases} m \mid m + a_1 \times h(k_1, I_1, r) + \ldots + a_n \times (h(k_1, I_1, r))^n & = & h(k_1, I_1, r+1) \\ m \mid m + a_1 \times h(k_2, I_2, r) + \ldots + a_n \times (h(k_2, I_2, r))^n & = & h(k_1, I_1, r+1) \\ \ldots & & \ldots \\ m \mid m + a_1 \times h(k_n, I_n, r) + \ldots + a_n \times (h(k_n, I_n, r))^n & = & h(k_n, I_n, r+1) \end{cases}$$

Let $f(x) = k + a_1 \times x + \ldots + a_n \times (x)^n$. The server broadcasts $r, T_s, f(1), f(2), \ldots, f(n)$.

Since the size of q (in $mod(q)$) is $O(\log n)$, given a message m of $O(\log n)$ bits, the overall length of the broadcast message has $O(n \log n)$ bits. In a typical point-to-point protocol using encryption, not only the same message has to be encrypted n times (with n different keys), the n pieces of ciphertext of a total of $O(n \log n)$ bits must also be sent. In some secure broadcast protocols (e.g., [4]), the broadcast message has $O(n^3 \log n)$ bits.

Nevertheless, in an environment such as where the recipients are scattered around in many corners of a large network, the savings in terms of efficiency in avoiding encryption may be outweight by the increased total network traffic and load.

5 Secure Communication

If the server can securely convey a secret to a client using polynomial interpolation, then intuitively the server can also securely communicate any message without using encryption or decryption. Our protocol for such secure communication is as follows. We call it NED-C, for Communication with No Encryption or Decryption.

Suppose A and B share a secret k. For A to send B a secret message m, A first selects a random number r and solves the following equation for a_1:

$$m + a_1 \times h(k, r) = h(k, r+1)$$

A then sends (r, x) to B, where $x = m + a_1 \times h(k, r+2)$. In other words, A sends to B

$$m + h(k, r+2) \times (h(k, r+1) - m)/h(k, r)$$

in additional to r. B then uses interpolation to compute m. A can include a nonce or a timestamp in the message to prove its freshness, as we describe in Section 2.4.

All security arguments for NED-A(2) apply. For example, a known-plaintext attack on this communication algorithm becomes a known-plaintext attack on the secure hash function $h()$, but of a more complicated form. Securing a one-way hash function is generally thought to be much easier than designing a secure encryption algorithm, at least in the practical sense. Also, r is a salt value [13] that should not be reused.

One difference between NED-C and NED-A(2) is that, if the hash function $h()$ is an onto mapping (also called a surjective function), and m does not contain redundancy, then an attacker who eavesdrops on x cannot obtain any information on m. This secrecy property is information theoretic because, for any m, there exists a k to satisfy the above equation.

To detect that the message has been modified during transmission, either m has sufficient redundancy or we can attach a secure checksum – for example, of the form $h(k, m, r, T)$, where T is a timestamp. In these cases, the secrecy argument becomes only computational.

Naturally, when m is long, we can divide it into blocks and encrypt them individually. One secure checksum for the entire message is sufficient. It is generally not a good idea to use the same r for every block, because of known-plaintext attacks. One solution is as follows. Suppose m can be divided into blocks as m_i, $i = 1, 2, \ldots, t$. We define the ciphertext for m_i to be:

$$m_i + (h(k, r+2, i) \times (h(k, r+1, i) - m_i)/h(k, r, i)$$

We can also use chaining methods. For example, we can change the ciphertext for m_i to be:

$$m_i + (h(k, r+2, c_{(i-1)}) \times (h(k, r+1, c_{(i-1)}) - m_i)/h(k, r, c_{(i-1)})$$

where $c_{(i-1)}$ is the ciphertext for block $m_{(i-1)}$. The initial c_0 can be a salt value. This method is commonly known as ciphertext feedback. Other modes of operation are possible but we do not discuss them here.

6 Related Work

The earliest authentication protocol we are aware of that does not use encryption is [7]. The protocols there use bit-wise exclusive-or operation and thus, compared with the NED-A protocols, must make more assumptions about the one-way hash functions, as we explain in Section 3.3.

Polynomial interpolation has been used in secret-sharing schemes [16], which deal mainly with the secrecy requirement. However, we are not aware of any previously proposed authentication protocol that is based on polynomial interpolation instead of encryption. Since we develop authentication and secure broadcast protocols, we must handle additional issues such as insider attacks, integrity, and authenticity. The possibility of doing the "reverse calculation" in polynomial interpolation, which is crucial for our new protocols, has been noted before (e.g., [8, Appendix B]). Other secret-sharing schemes where a similar "reverse calculation" can be done are also potentially applicable here.

We are not aware of any previous secure broadcast algorithms that do not use encryption. Some of the previous algorithms, such as Secure Broadcast Using Secure Lock [4], also need to associate one positive number with each of the n clients where the numbers must be relatively prime to each other. Managing the allocation of such numbers to a large group of clients – for example, all the hosts and users on the Internet – can be a serious problem in practice. Moreover, the arithmetic in Secure Broadcast Using Secure Lock must be computed in a field containing the product of all the n relatively prime numbers. Thus the size of field is at least $O(n^2 \log n)$ bits long [10, pp.9–10] and the broadcast message has $O(n^3 \log n)$ bits, whereas in NED-B protocols, the field size is only $O(\log n)$ and the broadcast message has $O(n \log n)$ bits.

7 Summary and Future Work

We describe Needham-Schroeder type authentication and key-distribution protocols that use polynomial interpolation instead of encryption. These protocols are simple, elegant, and efficient when compared with conventional protocols based on encryption. We also show how to use the same techniques for very efficient secure broadcast and secure end-to-end communication.

For future work, it is an open question whether the NED techniques can be beneficial when the clients register public keys instead of shared secrets with the server.

Acknowledgment

During a discussion on April 8, 1994, Professor D.J. Wheeler of the University of Cambridge pointed out the possibility of insider attack mentioned in Section 2.2. Peter G. Neumann of SRI has provided important technical and editorial comments on drafts of this paper.

References

[1] T.A. Berson, L. Gong, and T.M.A. Lomas. Secure, Keyed, and Collisionful Hash Functions. December 1993. Included in this technical report.

[2] B. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic Design of a Family of Attack-Resistant Authentication Protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679–693, June 1993.

[3] M. Burrows, M. Abadi, and R.M. Needham. A Logic for Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[4] G.H. Chiou and W.T. Chen. Secure Broadcasting Using Secure Lock. *IEEE Transactions on Software Engineering*, 15(8):929–934, August 1989.

[5] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–65, November 1976.

[6] W. Diffie and M.E. Hellman. Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *IEEE Computer*, 10(6):74–84, 1977.

[7] L. Gong. Using One-Way Functions for Authentication. *ACM Computer Communication Review*, 19(5):8–11, October 1989.

[8] L. Gong. *Cryptographic Protocols for Distributed Systems*. Phd dissertation, University of Cambridge, England, April 1990.

[9] L. Gong. On Efficient and Secure Broadcasting. March 1992. Included in Technical Report SRI-CSL-94-03, Computer Science Laboratory, SRI International, Menlo Park, California, March 1994.

[10] G.H. Hardy and E.M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, Oxford, England, 1979. First edition 1938, fifth edition 1979, reprinted (with corrections) 1983.

[11] D. Knuth. *The Art of Computer Programming, Vol.2: Seminumerical Algorithms*. Addison-Wesley, Reading, Massachusetts, 1969.

[12] R.C. Merkle. A Fast Software One Way Hash Function. *Journal of Cryptology*, 3(1):43–58, 1990.

[13] R. Morris and K. Thompson. Password Security: A Case History. *Communications of the ACM*, 22(11):594–597, November 1979.

[14] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[15] R.L. Rivest. The MD5 Message-Digest Algorithm. Request for Comments 1321, Internet Activities Board, April 1992.

[16] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[17] *Data Encryption Standard*. U.S. National Bureau of Standards, January 1977. U.S. Federal Information Processing Standards Publication, FIPS PUB 46.