New Protocols for Third-Party-Based Authentication and Secure Broadcast

Li Gong

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

Abstract. This paper gives Needham-Schroeder style third-party-based authentication and key-distribution protocols that do not use conventional cryptosystems such as DES. The new protocols use polynomial interpolation and secure keyed one-way hash functions, and are efficient compared with protocols using conventional cryptosystems. The same techniques are generalized to develop protocols for secure broadcast and for secure communication.

1 Authentication

Authentication, the process of identifying entities such as users and machines, is a crucial element in secure distributed systems and networks. Authentication is closely related to key distribution, and the two issues are often treated jointly [8, 19].

Research in third-party-based authentication protocols has been significantly influenced by the work of Needham and Schroeder [19], and 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 [10, 19]. Prior to authentication and session key distribution, a client either shares a secret key or (securely) registers a public key with the server [19]. In the rest of our discussion, we examine only the case in which server and clients share symmetric keys, and we use the term authentication to mean third-party-based authentication and key distribution.

In an effort to reduce the use of encryption in authentication protocols (partly to increase protocol efficiency), Gong suggests an authentication protocol that uses only secure keyed one-way hash functions [11]. 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 fashion from the secret key that S and A 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., [3]).

In this paper, we propose new authentication protocols that do not use a conventional cryptosystem such as DES [24]. Instead, the protocols are based on polynomial interpolation [16, pp.484-486], similar to Shamir's use in secret-sharing [23]. The same techniques can also be used for secure broadcast and secure communication, and 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. 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:

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

All computations are performed $mod(q)$ where q is a suitably large prime number [16]. 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]

The above procedure is like the "reverse calculation" of secret sharing in that, instead of calculating some arbitrary "shadows" of a given secret [23], 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.

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

A knows k_1, and thus can use interpolation to solve the following equations to obtain k [16]:

$$\left\{\begin{array}{rcl} k + a_1 \times 1 + a_2 \times (1)^2 & = & k_1 \\ k + a_1 \times 3 + a_2 \times (3)^2 & = & f(3) \\ k + a_1 \times 4 + a_2 \times (4)^2 & = & f(4) \end{array}\right. \quad (2)$$

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

$$\left\{\begin{array}{rcl} k + a_1 \times 2 + a_2 \times (2)^2 & = & k_2 \\ k + a_1 \times 3 + a_2 \times (3)^2 & = & f(3) \\ k + a_1 \times 4 + a_2 \times (4)^2 & = & f(4) \end{array}\right. \quad (3)$$

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 [23], and the guarantee of k's secrecy is information theoretic.

By the same argument, the attacker cannot compute k_1 or k_2. However, from the set of equations (1), we can easily deduce the relation that $k_1 - 3 \times k_2 = f(4) - 3 \times f(3)$. Thus, key distribution is limited to one session only, since after two sessions the attacker will be able to compute k_1 and k_2 using the above relation.

2.2 Prevention of Insider Attacks

In the preliminary protocol described in Section 2.1, an insider can attack more effectively. For example, after 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.[2]

One solution is to use a secure keyed one-way hash function. The following definition is adapted from [2], which also contains more references to works on one-way hash functions.[3]

Definition 1 *A function h() that maps a key and a second bit string to a string of a fixed length is a secure keyed one-way hash function if it satisfies the following five properties:*

1. *Given k and x, it is easy to compute h(k,x);*
2. *Given k and h(k,x), it is hard to compute x;*
3. *Without knowledge of k, it is hard to compute h(k,x) for any x;*
4. *Given k it is hard to find two values x and y such that h(k,x) = h(k,y), but $x \neq y$;*
5. *Given (possibly many) pairs x and h(k,x), it is hard to compute k.*

[2] This attack was first pointed out to me by Professor D.J. Wheeler of the University of Cambridge during a discussion on April 8, 1994. A reviewer of this conference later referred to me the work by Blakley et al. [5] where the problem of how to replenish new shadows in a secret-sharing scheme when pieces of old shadows are compromised was studied. That problem is related to the insider attack we consider here, though we have the additional (and significant) constraint that the shadows – the keys clients and the server share – are usually long term keys that do not or cannot get changed after each session of authentication.

[3] Paul Van Oorschot of Bell-Northern Research recently pointed out to me two papers [15, 21] where a definition of manipulation detection code is given. That definition is similar to but different from our definition of a secure keyed one-way hash function in that their definition does not require property 2 in ours and interprets hardness in property 3 specifically to be as hard as random guessing.

We note that, for a fixed key k, the secure keyed one-way hash function $h()$ behaves as a one-way hash function. However, it is unclear whether a one-way hash function such as MD-5 [22] can be securely used in a keyed fashion.

Given a secure keyed one-way hash function $h()$, the server selects a random number r and solves the following equations:

$$\left\{\begin{array}{rcl} k + a_1 \times 1 + a_2 \times (1)^2 & = & h(k_1, r) \\ k + a_1 \times 2 + a_2 \times (2)^2 & = & h(k_2, r) \end{array}\right. \quad (4)$$

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 $h(k_2, r)$.

Moreover, although A can compute $h(k_2, r)$, it 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 k_2 is well chosen (e.g., random, and from a sufficiently large space), 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. Moreover, r should not be used in multiple authentication sessions for the same clients, because after two sessions an outside attacker can use the relation $h(k_1, r) - 3 \times h(k_2, r) = f(4) - 3 \times f(3)$ to compute $h(k_1, r)$ and $h(k_2, r)$ and attack in future sessions.

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. 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 favoured 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.

A better protocol is to let the server solve the following equations:

$$\left\{\begin{array}{l} k + a_1 \times h(k_1, r) + a_2 \times (h(k_1, r))^2 = k_1 \\ k + a_1 \times h(k_2, r) + a_2 \times (h(k_2, r))^2 = k_2 \end{array}\right. \quad (5)$$

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 set of equations (5), S can select another value for r and repeat the process. Alternatively, we can add one more bit to the output of the hash function. For example, we can replace $h(k_1, r)$ and $h(k_2, r)$ in the above equations with $h(k_1, r) \mid 0$ and $h(k_2, r) \mid 1$, where "$x \mid y$" denotes x concatenated with y. This change guarantees that the equations have suitable solutions, even when k_1 happens to be equal to k_2.

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:

$$\begin{cases} k + a_1 \times h(k_1, r) + a_2 \times (h(k_1, r))^2 = h(k_1, r+1) \\ k + a_1 \times h(k_2, r) + a_2 \times (h(k_2, r))^2 = h(k_2, r+1) \end{cases} \quad (6)$$

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 in our setting where each client holds (i.e., shares with the server) only one secret that is long term (i.e., do not or cannot get changed frequently between authentication sessions) and small in size (e.g., 8 characters), and the session key is of a similarly small size. This is because otherwise for A to find out B's key k_2, it is just a matter of solving a higher order equation. This intuition is supported by the work of Blundo et al. [7] where they proved a theorem (for the non-interactive case of conference key distribution) that, in order to prevent a group of colluders from compromising others' secrets, a user needs to hold secret information that is many times the size of the session key. (The authors there gave a protocol based on the concept of *symmetric polynomials*.)

2.3 Integrity

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

In other applications, it is desirable 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, in which case the server can conveniently distribute $k \mid k$ instead of k. In other words, S solves the following equations:

$$\begin{cases} (k \mid k) + a_1 \times h(k_1, r) + a_2 \times (h(k_1, r))^2 = h(k_1, r+1) \\ (k \mid k) + a_1 \times h(k_2, r) + a_2 \times (h(k_2, r))^2 = h(k_2, r+1) \end{cases} \quad (7)$$

In this protocol, A (or B) solves the above equations as before, checks that in the result (i.e., $k \mid k$) 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 secure keyed 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. Here, notation $h(x_1, x_2, \ldots, x_n)$ denotes that $x_2, \ldots, x_n$ are concatenated to form the second input string to the hash function $h()$. After retrieving the session key, A can recompute this checksum and compare it with the received checksum. In order to forge a seemingly legitimate checksum, the attacker needs to know k_1. Similarly, S can send $h(k_2, k, r, B, A)$ to B.

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:

$$\begin{cases} (k \mid k) + a_1 \times h(k_1, f_{id1}, r) + a_2 \times (h(k_1, f_{id1}, r))^2 \\ \qquad = h(k_1, f_{id1}, r+1) \\ (k \mid k) + a_1 \times h(k_2, f_{id2}, r) + a_2 \times (h(k_2, f_{id2}, r))^2 \\ \qquad = h(k_2, f_{id2}, r+1) \end{cases} \quad (8)$$

Here, f_{id1} and f_{id2} can be identical (e.g., when they are the server's timestamp) or different (e.g., 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 instead send A an extra secure checksum, $h(k_1, f_{id1}, k, r, A, B)$, that is also fresh. (This checksum will also serve as a message integrity check.) If the server's timestamp is used, then the clients must know its exact value in order to compute the checksums. Thus the timestamp needs to be sent as plaintext in the server's message unless the clients can accurately predict the value of the timestamp.

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), where n is the number of clients. (The name NED is due to "historic" reasons, and A is for authentication.)

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 nonces 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 → B: I_a, I_b, N_a
2. B → S: I_a, I_b, N_a, N_b
3. S → B: $r, f(1), f(2), h(k_2, N_b, k, r, I_b, I_a)$
4. S → 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 session key k and a nonce r, and solves the following equations for a_1 and a_2:

$$\left\{\begin{array}{l} k + a_1 \times h(k_1, I_a, r) + a_2 \times (h(k_1, I_a, r))^2 \\ \qquad = h(k_1, I_a, r+1) \\ k + a_1 \times h(k_2, I_b, r) + a_2 \times (h(k_2, I_b, r))^2 \\ \qquad = h(k_2, I_b, r+1) \end{array}\right. \tag{9}$$

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 above 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 suitable 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. After receiving message 4, A uses interpolation to solve the following equations to obtain k.

$$\left\{\begin{array}{lcl} k + a_1 \times h(k_1, I_a, r) + a_2 \times (h(k_1, I_a, r))^2 & & \\ & = & h(k_1, I_a, r+1) \\ k + a_1 \times 1 + a_2 \times (1)^2 & = & f(1) \\ k + a_1 \times 2 + a_2 \times (2)^2 & = & f(2) \end{array}\right. \tag{10}$$

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, is fresh, and has not been modified during transmission. Therefore k is indeed the session key the server has distributed. Upon finding a mismatch, A terminates the protocol run or takes other appropriate actions. B's procedure 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, and a preliminary analysis using BAN-like logics [8] does not reveal any security problems. In an exhaustive cryptanalysis, the attacker may attempt to build up a dictionary of possible session keys and hash values. One way to increase the difficulty of this attack is to use longer texts in the input to the hash function. This method does not lose much in efficiency, because a hash function is fast to compute and the length of its output 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. 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, $i = 1, 2, \ldots, n$.

$$\left\{\begin{array}{l} k + a_1 \times h(k_1, I_1, r) + \ldots + a_n \times (h(k_1, I_1, r))^n \\ \qquad = \quad h(k_1, I_1, r+1) \\ k + a_1 \times h(k_2, I_2, r) + \ldots + a_n \times (h(k_2, I_2, r))^n \\ \qquad = \quad h(k_2, I_2, r+1) \\ \qquad \ldots \; = \; \ldots \\ k + a_1 \times h(k_n, I_n, r) + \ldots + a_n \times (h(k_n, I_n, r))^n \\ \qquad = \quad h(k_n, I_n, r+1) \end{array}\right. \tag{11}$$

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

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

$$\left\{\begin{array}{lcl} k + a_1 \times h(k_i, I_i, r) + \ldots + a_n \times (h(k_i, I_i, r))^n & & \\ & = & h(k_i, I_i, r+1) \\ k + a_1 \times 1 + \ldots + a_n \times (1)^n & = & f(1) \\ k + a_1 \times 2 + \ldots + a_n \times (2)^n & = & f(2) \\ \ldots & = & \ldots \\ k + a_1 \times n + \ldots + a_n \times (n)^n & = & f(n) \end{array}\right. \tag{12}$$

Note that the above equations having a solution requires $h(k_i, I_i, r)$ being out of the range of 1 to n. This can be satisfied in many ways. For example, the server can try different values of r.

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, is fresh, and has not been modified during transmission. Handshake messages can be added if required. The security arguments are similar to that for NED-A(2).

To eliminate the need for sending the n additional hash values, S can add redundancy by replacing k with $k \mid k$ and solving

$$\left\{\begin{array}{l} (k \mid k) + a_1 \times h(k_1, I_1, r) + \ldots + a_n \times (h(k_1, I_1, r))^n \\ \qquad = \quad h(k_1, I_1, I_2, \ldots, I_n, T_s, r, 1) \\ (k \mid k) + a_1 \times h(k_2, I_2, r) + \ldots + a_n \times (h(k_2, I_2, r))^n \\ \qquad = \quad h(k_2, I_1, I_2, \ldots, I_n, T_s, r, 2) \\ \qquad \ldots \; = \; \ldots \\ (k \mid k) + a_1 \times h(k_n, I_n, r) + \ldots + a_n \times (h(k_n, I_n, r))^n \\ \qquad = \quad h(k_n, I_1, I_2, \ldots, I_n, T_s, r, n) \end{array}\right. \tag{13}$$

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 equations for the clients to solve need to be changed accordingly. Because the use of $k \mid k$ provides redundancy for integrity check, the server needs to send only the following messages, which can be sent in a single broadcast:

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

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 Comparison with Related Work and Discussion

Key distribution protocols using a trusted server can be roughly classified into the following two types. One is by Blom [6] and others (e.g., [18, 13, 7]) in which the server distributes a large amount of keys to clients who later compute session or conference keys using the distributed keys together with the identities of those clients in the session. The theme in these protocols is not how the server distributes the keys to the clients, but rather how the clients use the keys to generate session keys. The NED protocols we propose in this paper are not of this type. Nevertheless, the work by Blundo et al. [7] is quite relevant in that they proved that, to prevent a group of colluders from compromising other clents' keys, the total size of the keys the server needs to distribute to each client must be many times of the size of a session key, not assuming the use of one-way hash functions.

The other type is the Needham-Schroeder style third-party-based authentication [19] where each client shares a small amount of (long term) secret with the server, who distributes a session key using the shared secrets to the clients upon request. The earliest work we are aware of on this type of protocols that do not use conventional cryptosystems is [11].

The protocols there (and elsewhere such as in [3]) use secure keyed one-way hash functions, and sometimes in bit-wise exclusive-or operations to protect data secrecy. This usage may implicitly require 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 [2, 21], and may also be hard to analyze. For example, given that $h()$ is a secure keyed 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. Depending on the definition of hardness, $g()$ may still be a secure keyed one-way hash function, although clearly $g()$ is not suitable for use in a bit-wise exclusive-or operation to protect data secrecy. The NED-A protocols in this paper do not require such statistical properties.

Polynomial interpolation has been used in secret-sharing schemes [23], which deal mainly with the secrecy requirement. For authentication and secure broadcast protocols, we must handle additional issues such as insider attacks, freshness, integrity, multiple sessions, and authenticity. We are not aware of any previously proposed third-party-based authentication and key distribution protocol that is based on polynomial interpolation instead of conventional encryption. The possibility of doing the "reverse calculation" in polynomial interpolation, which is crucial for our new protocols, has been noted before (e.g., [12, Appendix B]). Other secret-sharing schemes where a similar "reverse calculation" can be done are also potentially applicable here.

Compared with currently available third-party-based authentication protocols using conventional encryption, such as Kerberos [20], the NED-A protocols' main advantage is efficiency, though issues such as licensing and export control may also be relevant. To distribute a session key to n clients, the server and the clients each computes $n + c$ hash functions (where c is a small constant), and solves n equations with n unknowns, where the computational complexity of polynomial interpolation is $O(n(\log n)^2)$ [16]. In conventional protocols, the server does n encryptions and each client does at least one decryption. Since it is widely accepted that a one-way hash function is much (e.g., at least one to two orders of magnitudes [21]) cheaper to compute than a traditional encryption algorithm such as DES [24], the NED-A(n) protocols are more efficient when n is a small number such as 2. Note that the size of q (in $mod(q)$) needs to increase only logarithmically when n increases.

Other improvements to the NED-A protocols 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 here.

4 Secure Broadcast

If the server can distribute a session key to n clients, then intuitively the server can also broadcast a (secret) message to the n clients using the NED method. Before giving the details of our new protocol, we briefly review the problem of secure broadcast.

When S must send a message m to a number of destinations $P_i, i = 1, \ldots, n$, and the message content must be kept private for the intended recipients, the message is usually encrypted before being broadcast. Depending on the pattern of key sharing among communicating parties, different protocols can be used. For example, S can encrypt m with a key S and P_i share (e.g., k_i) and send the ciphertext to P_i. This amounts to a total of n encryptions, and n point-to-point messages or a broadcast of n pieces of ciphertext. On the other hand, if S and the P_i's share a group key, then one encryption and one broadcast of a single ciphertext is sufficient. However, since S may need to send messages to different sets of recipients, S must maintain many such group keys, up to $O(2^n)$.

Suppose that each server or client has a pair of public and private keys [10] and that each party knows the public key of every other party, then S can choose an encryption key (e.g., for use in DES) at random and encrypt m. S then uses its private key to sign a timestamp (to defend against replay attacks), the DES key, and a checksum of the original message (e.g., a one-way hash function of m), and encrypts this signature with each client's public key. Finally, S broadcast (or multicast) the n encrypted signatures and the encrypted message. At the receiving end, each client can decrypt a suitably encrypted signature, obtain the DES key (using the S's public key), and then retrieve the message. Clearly, anyone other than the clients will not be able to retrieve m. If the secrecy of the message is not required, then S can simply broadcast m together with, for example, m's checksum signed with S's private key.

The problem we examine here is how to securely broadcast a message when public-key systems are not used, and how to avoid encrypting the message n times or maintaining $O(2^n)$ group keys.

4.1 Common Attacks in Secure Broadcast

Some authors of secure broadcast protocols (e.g., [9, 1]) focus their attention mostly on attacks on secrecy, and do not discuss other attacks, for example, the insider attack in Section 2.2. We now highlight the risks with a case study of the Chiou-Chen protocol [9]. We discuss these attacks and also suggest counter measures.

The Chiou-Chen protocol works as follows. Assume that S and P_i share a key k_i. S selects a random encryption key k, encrypts m with it, and computes $\{k\}_k$. Here, $\{m\}_k$ denotes m encrypted with key k. Then S solves the following equations for x using the Chinese Remainder Theorem.

$$\left\{ \begin{array}{lcll} x & \equiv & \{k\}_{k_1} & mod \quad N_1 \\ & \dots & & \\ x & \equiv & \{k\}_{k_n} & mod \quad N_n \end{array} \right. \tag{14}$$

Here, N_i is a publicly known number associated with P_i. These numbers must satisfy a few conditions, e.g., they must be pairwise prime. S now broadcasts $(S, x, \{k\}_k, \{m\}_k)$.

After receiving the broadcast, P_i computes $\{k\}_{k_i}$ (from x and N_i) and decrypts it with k_i to obtain k, with which P_i decrypts $\{k\}_k$. If the last decryption does not reveal k, it means either that the message has been modified during transmission (the integrity of $\{m\}_k$ must be checked separately) or that the broadcast is not intended for P_i. Otherwise, P_i uses k to decrypt $\{m\}_k$, thus obtaining m. Note that m is only encrypted once, and only one piece of ciphertext is broadcast.

The protocol implicitly requires strong (and possibly impractical) assumptions of the protocol execution environment. When these assumptions do not hold, the protocol is vulnerable to replay and masquerading attacks, especially by a legitimate but malicious recipient of a broadcast message. For example, anyone can record S's message and play it back again. A recipient cannot detect a replay except by keeping a record of past messages and keys, thus noticing message repetitions. If recipients cannot be expected to keep such records, establishing the freshness of a message by other means is essential. For the sake of discussion, we assume that clocks are securely synchronized. (Other methods to ensure freshness, such as challenge and response, can also be used.) In this case, a usual solution is to attach to m a timestamp t to indicate the time of message creation, i.e., replacing m with (t, m) in the protocol.

However, an insider (say P_1) can still attack. Being a legitimate recipient, P_1 knows k after receiving the broadcast. Thus P_1 can replace (t, m) with (t', m'), where t' is a current timestamp and m' is an arbitrary message, and then rebroadcast the message. If other recipients cannot detect that k is reused and there is no other hardware or software mechanism to identify the message origin, the new broadcast message will be taken as from S rather than from P_1, with potentially serious consequences. Timestamping $\{k\}_k$ by broadcasting $(S, x, \{t, k\}_k, \{m\}_k)$ still has the same weakness because P_1 who knows k can easily forge a seemingly legitimate $\{t', k\}_k$ with a new timestamp t'. Note that the victims are confined to recipients of the original broadcast.

One solution to the above problems is to timestamp key k by modifying the equations to be $x \equiv \{t, k\}_{k_i} \bmod N_i$. This is clearly better because t can specify k's creation time as well as its lifetime. Since P_1 does not know k_i, $i \neq 1$, P_1 cannot forge $\{t, k\}_{k_i}$. Thus P_1 cannot convince others that an expired key k is still valid, and a future message broadcast by P_1 using k will be rejected.

There is yet another problem that is not solved by this new version. In many systems, for example, the Internet, a broadcast channel is a simulation. Sometimes such a channel will behave correctly only if all parties concerned follow the protocol (e.g., [4]). In such an environment, it is likely that P_1 can interfere with the broadcast before another recipient receives the message. In particular, P_1 can intercept the broadcast message m from S, change its content to m', and then send it on to other recipients before k expires. This type of attack is often possible when the message delivery system is unreliable or slow, and especially when broadcast is simulated in software instead of implemented in hardware.

One protection against this attack is to omit the intermediate key k altogether, solve $x \equiv \{t, m\}_{k_i} \bmod N_i$, and broadcast (S, x). This change makes the protocol secure against described replay or masquerading attacks by outsiders as well as by insiders, since no one besides S and P_i can create $\{t, m\}_{k_i}$. (Note that $\{t, m\}_{k_i}$ can be broken into smaller blocks if necessary.) The efficiency of this version is not as good as the original Chiou-Chen protocol, however, because m, which may be significantly longer than k, is now encrypted n times. Nevertheless, only one piece of ciphertext is broadcast, and its length is roughly the same as that in the original protocol.

This new version can also be slightly modified to broadcast different private messages to multiple destinations. For example, suppose S wants to send m_i privately to P_i. Instead of sending n separate messages, S can solve equations $x \equiv \{t, m_i\}_{k_i} \bmod N_i$ for x, and then simply broadcast x.

Suppose insider attacks must be considered, and yet not everyone can be expected to store past messages, and there is no hardware or software support to ensure that a broadcast cannot be interfered with before the message reaches all destinations. We now propose a new protocol that achieves the level of efficiency of the original Chiou-Chen protocol. Suppose $h()$ is a one-way hash function. Server S selects k and encrypts m to obtain $\{m\}_k$. S then solves the following set of equations for x and broadcast $(S, x, \{m\}_k)$.

$$\left\{ \begin{array}{lcll} x & \equiv & \{t, k, h(m)\}_{k_1} & mod \quad N_1 \\ & \dots & & \\ x & \equiv & \{t, k, h(m)\}_{k_n} & mod \quad N_n \end{array} \right. \tag{15}$$

Here $h(m)$ acts as a checksum of m. Recipient P_i obtains m as before, computes $h(m)$, and compares it with the checksum value received from S. A discrepancy suggests accidental modification or deliberate tampering of the broadcast message.

Since the original Chiou-Chen protocol is secure against known-plaintext attacks [9], and the new protocol is different only in that two more data items – a timestamp t and a checksum $h(m)$ – are encrypted, the new protocol is also likely to be resistant to known-plaintext attacks. Moreover, since the attacker does not know k_i and thus cannot modify $\{t, k, h(m)\}_{k_i}$ to forge a seemingly legitimate checksum, any illegal modification of the broadcast message will be detected by P_i. This protocol's efficiency is close to that of the original protocol because m is encrypted once, only one piece of ciphertext is broadcast, the computation of a one-way hash function can be very efficient, and the checksum adds only a few extra bytes. Note that, unlike the Chiou-Chen protocol, integrity of m already can be checked by $h(m)$ and need not be protected by encryption or a separate checksum.

If $h()$ is a secure keyed one-way hash function, then the amount of encryption and the length of the broadcast message can be further reduced by replacing $\{t, k, h(m)\}_{k_i}$ with $(t, h(k_i, m, t))$ in the set of equations (15). S now broadcasts $(S, x, \{m\}_k, t)$. After obtaining m by decrypting $\{m\}_k$, P_i can recompute $h(k_i, m, t)$ and compare it with the received value to check message integrity. In fact, if the privacy of the broadcast message is unimportant, no encryption is needed. S just broadcasts (S, x, m, t).

4.2 A New Protocol NED-B

Returning to the NED techniques, we can broadcast by using the NED-A(n) protocol in Section 3.2 and replacing 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, where B is for broadcast. The server solves the following equations:

$$\begin{cases} m \mid m + a_1 \times h(k_1, I_1, r) + \ldots + a_n \times (h(k_1, I_1, r))^n \\ \qquad = \quad h(k_1, I_1, T_s, r, 1) \\ m \mid m + a_1 \times h(k_2, I_2, r) + \ldots + a_n \times (h(k_2, I_2, r))^n \\ \qquad = \quad h(k_2, I_2, T_s, r, 2) \\ \qquad \ldots = \ldots \\ m \mid m + a_1 \times h(k_n, I_n, r) + \ldots + a_n \times (h(k_n, I_n, r))^n \\ \qquad = \quad h(k_n, I_n, T_s, r, n) \end{cases} \tag{16}$$

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

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 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 conventional encryption may be outweight by the increased total network traffic and load.

4.3 Related Work in Secure Broadcast

There are a number of existing protocols for secure broadcast, and some protocols for conference key distribution can also be extended for this purpose (e.g., [9, 17, 1]). The three protocols cited here are typical in that the main concern is to defeat those attacks directly on message secrecy, but not other attacks such as by insiders or on message freshness.

Among these protocols, the closest to the NED-B protocol is perhaps the work by Berkovits [1] where the proposal uses polynomial interpolation but in a different form.

Another closely related protocol is by Chiou and Chen [9] since polynomial interpolation is a special case of the Chinese Remainder Theorem. Their protocol needs 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, hosts and users on the Internet – can be a serious problem in practice. Moreover, the arithmetic in their protocol must be computed in a field containing the product of all the n relatively prime numbers. Since the range containing n prime numbers is roughly $[1, n \log n]$ [14, pp.9–10], the size of the field is at least $O(n \log(n \log n)) = O(n \log n)$ bits long and the broadcast message has $O(n^2 \log n)$ bits, whereas in NED-B protocols, the field size is only $O(\log n)$ and the broadcast message has $O(n \log n)$ bits.

5 Secure Communication

If the server can securely convey a secret session key to a client using polynomial interpolation, then intuitively the server can also securely communicate any message to the client without using a conventional cryptosystem. One protocol for such secure communication is as follows. The protocol is called NED-C, where C is for communication.

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:

$$m + a_1 \times h(k, r) = h(k, r + 1) \tag{17}$$

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

$$\text{A} \rightarrow B\text{: } r, m + h(k, r + 2) \times (h(k, r + 1) - m)/h(k, r)$$

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 keyed one-way hash function $h()$, but of a more complicated form. Here r is a "salt value" that should not be reused to encrypt multiple messages. The inclusion of a salt value also causes the ciphertext to be slightly longer than the plaintext,

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 suitable k to satisfy equation (17).

To detect that the message has been modified during transmission, either m has sufficient redundancy or we can attach a secure checksum that, for example, can be 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, as m_i, $i = 1, 2, \ldots, w$, and "encrypt" them individually. One secure checksum for the entire message is sufficient. It is generally better not to use the same r for every block, because of known-plaintext and slicing attacks. We define the "ciphertext" for m_i to be c_i:

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

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

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

The initial block c_0 can be a salt value. This method is commonly known as ciphertext feedback. Other modes of operation are also possible.

6 Summary and Future Work

We propose Needham-Schroeder type authentication and key-distribution protocols that use polynomial interpolation and secure keyed one-way hash functions. Compared with protocols using conventional cryptosystems, the new protocols are quite efficient. We also show how to use the same techniques for secure broadcast and secure end-to-end communication.

For future work, it may be instructive to attempt reducing the use of secure keyed one-way hash functions in the NED approach. Also, it is unclear whether the NED techniques can be beneficial when clients do not share secrets with the server but instead register their public keys.

Acknowledgement

This paper contains most of the materials (in revised and expanded forms) in two of my earlier technical reports:

- "Authentication, Key Distribution, and Secure Broadcast in Computer Networks Using No Encryption or Decryption", SRI-CSL-94-08, May 1994.
- "On Efficient and Secure Broadcasting", March 1992, revised June 1993 and February 1994. Included in SRI-CSL-94-03, March 1994.

Many colleagues read the two reports (and some earlier drafts) and provided valuable feedback. The anonymous reviewers of this conference brought to my attention several references [1, 5, 7, 17], thus giving me the opportunity to make the presentation in this paper more complete.

References

[1] S. Berkovits. How To Broadcast a Secret. In *Advances in Cryptology: Proceedings of Eurocrypt '91*, volume 547 of *Lecture Notes in Computer Science*, pages 535–541. Springer-Verlag, New York, April 1991.

[2] T.A. Berson, L. Gong, and T.M.A. Lomas. Secure, Keyed, and Collisionful Hash Functions. December 1993. Included in Technical Report SRI-CSL-94-08, Computer Science Laboratory, SRI International, Menlo Park, California, May 1994.

[3] 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.

[4] K.P. Birman and T.A. Joseph. Reliable Communication in the Presence of Failures. *ACM Transactions on Computing Systems*, 5(1):47–76, February 1987.

[5] B. Blakley, G.R. Blakley, A.H. Chan, and J.L. Massey. Threshold Schemes with Disenrollment. In *Advances in Cryptology: Proceedings of Crypto '92*, volume 740 of *Lecture Notes in Computer Science*, pages 540–548. Springer-Verlag, New York, August 1992.

[6] R. Blom. An Optimal Class of Symmetric Key Generation Systems. In *Advances in Cryptology: Proceedings of Eurocrypt '84*, volume 209 of *Lecture Notes in Computer Science*, pages 335–338. Springer-Verlag, New York, April 1984.

[7] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Young. Perfectly-Secure Key Distribution for Dynamic Conferences. In *Advances in Cryptology: Proceedings of Crypto '92*, volume 740 of *Lecture Notes in Computer Science*, pages 471–486. Springer-Verlag, New York, August 1992.

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

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

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

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

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

[13] L. Gong and D.J. Wheeler. A Matrix Key Distribution Scheme. *Journal of Cryptology*, 2(2):51–59, 1990. Springer-Verlag.

[14] 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.

[15] R.R. Jueneman, S.M. Matyas, and C.H. Meyer. Message Authentication with Manipulation Detection Codes. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 33–54, Oakland, California, April 1983.

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

[17] C.S. Laih, J.Y. Lee, and L. Harn. A New Threshold Scheme and its Application in Designing the Conference Key Distribution Cryptosystem. *Information Processing Letters*, 32:95–99, August 1989.

[18] T. Matsumoto and H. Imai. On the Key Predistribution System: A Practical Solution to the Key Distribution Problem. In *Advances in Cryptology: Proceedings of Crypto '87*, volume 293 of *Lecture Notes in Computer Scien*, pages 185–193. Springer-Verlag, New York, October 1987.

[19] 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.

[20] B.C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9), September 1994.

[21] B. Preneel. Cryptographic Hash Functions. *European Transactions on Telecommunications*, August 1994. To appear.

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

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

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