New Classes and Applications of Hash Functions

Mark N. Wegman and J. Lawrence Carter*

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Abstract: In this paper we exhibit several new classes of hash functions with certain desirable properties, and introduce two novel applications for hashing which make use of these functions.

One class of functions is small, yet is almost universal$_2$. If the functions hash n-bit long names into m-bit indices, then specifying a member of the class requires only $O((m + \log_2\log_2(n))\log_2(n))$ bits as compared to $O(n)$ bits for earlier techniques. For long names, this is about a factor of m larger than the lower bound of $m+\log_2 n-\log_2 m$ bits. An application of this class is a provably secure authentication techniques for sending messages over insecure lines.

A second class of functions satisfies a much stronger property than universal$_2$. We present the application of testing sets for equality.

The authentication technique allows the receiver to be certain that a message is genuine. An 'enemy' - even one with infinite computer resources - cannot forge or modify a message without detection.

The set equality technique allows the the operations 'add member to set', 'delete member from set' and 'test two sets for equality' to be performed in expected constant time and with less than a specified probability of error.

0. Introduction:

A hash function may be viewed as a means of assigning an abbreviation to a name. While hash functions are primarily used to implement associative memories, in this paper we explore some other uses for abbreviations. These new applications have motivated finding classes of hash functions with certain properties.

* J. L. Carter's current address: Department of Computer Science, Penn State University, University Park, PA 16802.

Use of a universal$_2$ class (or set) of hash functions has the advantage that good expected performance is assured for any input [CW]. In section 1 we strengthen the notion of a universal$_2$ set of hash functions to strongly universal$_n$ and strongly universal$_\omega$.

Known universal classes of hash functions are fairly large. If the functions in one of these classes can hash n-bit names, then the class typically contains $2^{O(n)}$ functions. Thus, $O(n)$ bits are required to specify a randomly chosen function. In Section 3, we present a set of functions which is 'almost' strongly universal$_2$ and is much smaller - only $\log(n)$ bits are required. This improvement can make some applications of hashing practical, for instance the authentication technique described below. Also, if used in conjunction with the extendible hashing scheme of [FNPS], one can make a fast, practical and completely general associative memory subroutine package.

A possibly important use of these functions, described in section 2, is a provably secure authentication system. This system allows the receiver of a message to be sure of the authenticity of a message - that the message was not forged or modified by an unauthorized 'enemy'. It is necessary for the sender and the receiver to share a secret key whose length is on the order of the log of the length of the message. But unlike digital signatures based on public key cryptosystems, it can be proven that this system is secure even against an enemy with infinite computing power. Also, there are no messages which just happen to be easy to forge.

Section 4 gives a refinement of the authentication system which allows many messages to be sent using the same secret key, with each message requiring an additional but shorter key. The total length of the keys required for sending multiple messages asymptotically achieves the lower bound.

The application which motivated defining strongly universal$_\omega$ classes of functions - a set equality tester - is given in Section 5. Assuming that sets are constructed using certain operations, we give a technique which has an input independent and small chance of error for determining that sets are equal. The expected running time of the algorithm is input independent and is linear in the number of construction operations and equality tests. (This is the only algorithm we have seen with the dubious distinction of requiring probabilistic analyses of both kinds - it can both make a mistake and take a long time doing so. Using a theorem of Gill, one can always make the running time constant at the expense of a larger probability of error.) Finally, in Section 6, we show how to construct a strongly universal$_\omega$ set of functions. The functions in this set can be evaluated rapidly.

1. Strongly Universal Sets of Hash Functions

To be universal$_2$, a set of functions from A to B must only satisfy a requirement on the probability that a randomly chosen function will map two points of A to the same value. For a set of functions to be universal$_n$, a randomly chosen function must, with equal probability, map any n distinct points of A to any n values in B; in other words, any n points must be distributed randomly throughout B by the functions. More formally,

Def: Suppose H is a set of hash functions, each element of H being a function from A to B. H is strongly universal$_n$ if given any n distinct elements $a_1,\ldots,a_n$ of A and any n elements $b_1,\ldots,b_n$ of B, then $|H|/(|B|^n)$ functions take a_1 to b_1, a_2 to b_2, etc. ($|X|$ means the number of elements in the set X.) A set of hash functions is strongly universal$_\omega$ if it is strongly universal$_n$ for all values of n.

Carter & Wegman [CW] suggest several classes of hash functions which turn out to be strongly universal$_2$, here we generalize one of these classes. Strongly universal$_n$ sets of functions can be created using polynomials over finite fields. In particular, let A and B both be the same finite field. Let H be the class of polynomials of degree less than n. H is strongly universal$_n$ since given any n distinct elements of A and corresponding elements of B, there is exactly one polynomial of degree less than n which "interpolates" through the designated pairs. (The standard linear algebra proof using the invertibility of the Vandermonde matrix also works with finite fields.)

It may seem peculiar to define a set of hash functions with A and B being the same size. However it is easy to make B smaller by, for instance, just choosing the last bits of the hashed value. If the size of the field is a power of two, the result will still be a strongly universal$_n$ class of functions; otherwise, it will still be 'close'.

In the last section, we will exhibit a strongly universal$_\omega$ set of functions.

2. Digital Signatures and Authentication Tags

It is often desirable to be able to send a message over an insecure line and yet allow the receiver to be certain of the identity of the sender. The post office may be viewed as an insecure line; the classical solution is to add a handwritten signature. The hope is that this written signature can be compared by the receiver to what he knows is a proper signature. Moreover, someone who intercepts the message cannot cut off the signature and paste it to a different document without detection. We seek to gain these advantages in the case of digital messages.

A written signature serves a number of functions:
1) It assures the receiver that the message was sent by an authorized person,
2) it can be used by the receiver to prove that the sender or someone authorized by him sent the message,

and
3) it identifies the sender to be the actual person indicated, not merely someone authorized by the person. Digital signatures as discussed by Diffie and Hellman [Diffie] serve the first two functions, while the authentication tags discussed here serve only the first. However, we will see that if there is a universally trusted person then authentication tags can also serve the second function. Neither authentication tags nor digital signature can serve the third function since the device which does the signing can be duplicated or given to a third party.

A digital signature consists of a string of bits which is concatenated with a message. This signature is a function of the message, but from the message and the signature it should be difficult if not impossible to determine the function. A checking function is published and available to all. This checking function allows anyone to test whether a signature is a valid signature for the particular message. Moreover, without the signing function (but even with the checking function) it is difficult to determine the correct signature to any alternate message. For all messages there exists a valid signature.

Diffie & Hellman [Diffie], and Rivest, Shamir, & Adleman [Rivest] have presented 'public key cryptosystems' which allow the above (in addition to allowing an interesting type of encryption.) However,

Theorem: No public key cryptosystem is unbreakable. That is, an enemy with unbounded computing resources can forge messages.
Proof: An enemy with enough time can guess all possible signatures for a particular message and when a valid signature is found use it.

Thus, all such signature schemes can be cracked in NP time, and in fact none have so far been shown NP-Complete. In fact, Adleman[Adleman] and Brassard[Brassard] have presented evidence that no public key cryptosystem can be NP-complete.

An authentication system works as follows: There is a set M of possible messages to be sent, and a set T of authentication tags. For instance, M might be the set of all character strings of length 10,000 or less, and T might be the set of bit strings of length 100. There is also a (publicly known) set of functions F, where each function in F maps M into T. To use the system, the sender and receiver secretly agree upon a 'key' which specifies one of the functions f in F. When the sender transmits a message m in M, he also sends the authentication tag f(m). The receiver checks that f applied to the message he received is indeed the tag he received. If so, he has some assurance the received message is not a forgery.

Since a forger may intercept a message and its valid tag, it must be impossible to find the function from the message and tag. Otherwise the forger might prevent the message from being sent and replace it with one of his own. In fact, knowing the value of f on one message must give no information about the value of f on any other message. We will show a little later how this can be accomplished.

An authentication tag differs from a signature in that the potential receiver of a message can also create the authentication tag and thus a proper message. Thus, the receiver cannot demonstrate to a third party that the message was actually transmitted and received, since the third party may think the receiver created the message. However, if there is a universally trusted agency, then authentication tags can be used to establish the authorship of messages. This works as follows: Each individual person X shares his secret function f_x only with the agency. To send a message to person Q, person P sends his message (tagged by his function) to the agency. The message must contain the name of the sender and receiver. The agency first verifies that the message it receives has the correct tag for the sender named in the message. Then it stores a copy, appends the tag via f_Q to the message, and forwards it on to Q. When Q verifies that the tag is correct, he accepts the message. Q can now prove to anyone who trusts the agency that P sent him the message by simply asking the agency to check its records.

A second possible disadvantage of authentication tags is that only a finite number of messages can be sent using a particular function. We will prove that any unbreakable scheme can only be used a finite number of times, with that number dependent on the size of the key and the desired probability of guessing the correct tag. We will also show that our scheme approaches the theoretical bound on the minimum key size needed to send a given number of messages with a desired level of security.

To compensate for these disadvantage, we can construct an authentication scheme which is provably unbreakable. That is to say no resources other than knowledge of the key allow an enemy to find the correct tag for a forged message. As the length of the tag grows the likelihood of a correct tag being appended to a message by an enemy, without knowledge of the key, becomes more and more remote.

To make this more precise, we will say that an authentication system is *unbreakable with certainty p* if after a function f is randomly chosen and after the forger is given any message m and the corresponding tag f(m), the forger cannot find a different message m′ for which he has better than a probability of p of guessing the correct tag. Note that this definition must hold for any m, even one chosen by the forger.

To create an authentication system which is unbreakable with certainty p, we can simply choose T to have at least 1/p elements, and let F be a strongly universal$_2$ class of hash functions from M to T. If we let H′ be the subset of H which maps m to f(m), we see that the only information that the forger has available to him is that the secret function is one of the functions in H′. However, the definition of strongly universal$_2$ implies that for any m′ distinct from m, the proportion of functions in H′ which map m′ to any particular tag t′ is $1/|T|$. Since $|T| \geq 1/p$, any choice the forger makes has no more than a probability of p of being correct.

[GMS] have found rather complicated strongly universal$_2$ sets of functions for exactly this purpose. The difficulty with their set and with other previously known strongly universal$_2$ sets is that the set of functions is so large that specifying a function in the class requires a key at least as long as the original message. It is desirable to use a key considerably shorter than the message. A second problem is that only one message per key can be sent, since knowledge of two message-tag pairs may give some information about the value of the function on some third message. We will solve these problems separately.

3. A Small, Almost Strongly Universal$_2$ Class

We wish to construct a set of hash functions from some large space A′ to a space B′. In the above, A′ is the set of messages and B′ is the set of possible tags. Let a′ and b′ be the length of the messages and tags, respectively. Let $s = b' + \log_2 \log_2(a')$. Let H be some strongly universal$_2$ class of functions which map bit strings of length 2s to ones of length s. The multiplicative scheme of [CW] is fine for this purpose. Each member of H′ will be constructed from a sequence of length $\log_2 a' - \log_2 b'$ of members of H. Suppose $f_1, f_2, \ldots$ is some such sequence. We will specify how to apply the associated member, f′ of H′ to a message. The message is broken into substrings of length 2s. If necessary, the last substring should be padded with blanks. Thus, if the message has n bits it will be broken into $\lceil \frac{a'}{2s} \rceil$ substrings. f_1 is applied to all the substrings and the resulting substrings are concatenated. By concatenating the resulting substrings, we have obtained a string whose length is roughly half the originals string's length. This process is repeated using $f_2, f_3, \ldots$ until only one substring of length s is left. The tag (i.e. the result of the hash function f′) is the low-order b′ bits of this substring. The key needed to specify f′ is the concatenation of the keys needed to specify $f_1, f_2, \ldots$. The multiplicative scheme suggested in [CW] has a key roughly twice the size of the input. If this class is used for H, the size of the key for H′ will be 4 s $\log_2(a')$. Thus, the key is roughly 4 times the length of the tags times the log of the length of the message. By comparison, the multiplicative scheme by itself would have a key whose length was twice that of the message.

Observe that assuming the functions in H can be evaluated in time proportional to s, the functions in H′ also can be evaluated in time proportional to the length of the message.

The sense in which H′ is 'almost' strongly universal_2 is given in the following theorem.

Theorem: Given any two distinct messages m_1 and m_2 and any two tag values t_1 and t_2, the number of functions which take m_1 to t_1 is $1/|B'|$ times the total number of functions. However, less than $2/|B'|$ of these functions will also take m_2 to t_2.

Proof sketch: Each time we halve the length of the messages, there is a small $(1/(2^s))$ chance that the two resulting strings are now identical. Since we iterate the halving process $\log_2 a'$-$\log_2 b'$ times, the chance that the two strings are identical at the next to last step is less than $\log_2 a'/(2^s)$, which is equal to $1/(2^{b'})$. Now the fact that function that does the last reduction is chosen from a strongly universal_2 class can be used to show that m_1 will be taken into any tag with equal probability, and as long as the penultimate strings were different, m_2 will also be taken into any string with probability equal to $1/|B'|$. Thus, if $t_1 \neq t_2$, then less than $1/|B'|$ of the functions will take m_2 to t_2, and otherwise, less than $2/|B'|$ will.

The above theorem can be contrasted with the definition of strongly universal_2, which says that $1/|B'|$ of the functions must take m_1 to t_1, and that $1/|B'|$ of these functions will also take m_2 to t_2. In terms of the authentication scheme, the theorem states after the enemy knows one message-tag pair, he can do no better than to find another message-tag pair which has probability $2/|B'|$ of being correct. Thus, the scheme is unbreakable with certainty $2/|B'|$, and this certainty can be made smaller than any predetermined value.

4. Authenticating Multiple Messages

The above method does not allow us to tag more than one message using the same function, since once the enemy knows two message-tag pairs, he may be able to determine more such pairs. One way around this problem might be to use a universal_n function, which would allow us to send n-1 messages, but a better method is as follows: There is a strongly universal_2 set F of functions from M to B, where B is the set of bit strings of length k. Each message in M must contain a message number between 1 and n. The secret key shared by the sender and receiver now consists of two parts. The first part specifies a function f in F. The second part of the key is a sequence $(b_1, \ldots, b_n)$ of elements of B. The sender must be certain never to send two messages with the same message number. To create the authentication tag t_i for the message m_i (a message with message number i) the sender first calculates $f(m_i)$ and then exclusive-or's this result with b_i. Since each message contains a message number, the receiver can duplicate this process to verify the tag is correct. (If a message is unnumbered, it is automatically rejected as a forgery.) We wish to show that this scheme is unbreakable with certainty $1/(2^k)$.

Theorem: Suppose some key $(f,(b_1,\ldots,b_n))$ has been chosen randomly from the set of keys. Let $m_1,\ldots,m_n$ be any n messages, with the restriction that the message numbers must all be different (we assume that m_i has number i.) Suppose a forger knows only the set F and the set of messages and their corresponding tags $t_i = f(m_i) \oplus b_i$. (We use $\oplus$ to denote the exclusive-or operation.) Then there is no new message (with any message number) for which the forger has a better than $1/(2^k)$ chance of correctly guessing the tag.

Proof: Suppose the forger wishes to guess the tag to the new message m. Without loss of generality, we assume m has the message number 1. For each t in B, define

$S_t = \{(g,b) \mid g \in F, b \in B, g(m_1) \oplus b = t_1 \text{ and } g(m) \oplus b = t\}$.

In other words, S_t is the set of partial keys (partial since only the first of n elements of b are specified) which are consistent with the fact that m_1 has tag t_1, and which give the bogus message m the tag t. It isn't hard to show that since F is strongly universal_2, each of the S_t's have the same size. Further, there is exactly one way to extend each partial key in S_t to a complete key which also assigns tag t_i to message m_i for $i=2,...,n$ (namely let $b_i = g(m_i) \oplus t_i$.) Thus, of all the keys which are consistent with the information which the forger has available, as many will assign to m any one tag as any other tag. Thus, the forger's probability of guessing the correct tag for m is $1/(2^k)$.

A similar theorem holds when you use an almost strongly universal_2 class.

We now summarize a proof that the number of bits required for the key by this scheme asymptotically approaches the optimal. A similar theorem has also been proved by V. Fak [Fak]. Rather than restricting our attention to schemes in which the messages must have a message number, we sketch a slightly more general scenario than in the previous theorem. Suppose a function has been selected from a set F. The forger chooses a message m_1 and tries to guess the correct tag. He is then told the correct tag t_1. Now the forger selects a second message m_2, trys to guess the tag, and then is told the correct tag t_2. This process is repeated n times. If we wish, we may require the forger to choose each message from a restricted subset of the set of all messages, or we may even have a fixed sequence of messages - these variations don't affect the following theorem.

Theorem: In the above scenario, if the forger's probability of success on his i-th guess is $\leq p_i$, then F must contain at least $1/(p_1 p_2 \dots p_n)$ functions.

Proof: Let $F_0 = F$ and $F_k = \{f \in F \mid f(m_i)=t_i$ for $i=1,...,k\}$. The forger might use the following strategy in his guessing: After choosing the i-th message, he enumerates the set F_{i-1}, randomly chooses a member of it, and guesses the tag $f(m_i)$. Since this has $\leq p_i$ chance of success, it must be the case that $|\{f \in F_{i-1} \text{ s.t. } f(m_i)=t_i\}| \leq p_i |F_{i-1}|$. The set on the left hand side is F_i, so we have $|F_{i-1}| \geq (1/p_i)|F_i|$. This is true for each i, so we have $|F_0| \geq (1/p_1)\dots(1/p_n)|F_n|$. The theorem follows since $F_0=F$ and $|F_n| \geq 1$.

Corollary: It requires at least $\sum_{i=1}^{n} -\log_2(p_i)$ bits of information to specify a randomly chosen member of F.

5. Testing Set Equality

In this section we present a linear algorithm for testing many sets for equality. More formally, suppose we have a sequence of requests which may name an arbitrary number of sets and an arbitrary number of elements. Each request can be one of the following three commands: add element x to set S, delete element x from set S, and test sets S_1, S_2 for equality. We can process a sequence of requests in expected time linear in the number of requests but with error probability ϵ times the number of requests. In addition to the above three, requests of one (but not more than one) of the three following forms may be included in the sequence: assign set S_1 to be the symmetric difference of S_2 and S_3, assign set S_1 to be the multi-set union of S_2 and S_3, or test element x for membership in set S.

The technique we will use is a modification of a known heuristic (see Schwartz). Some of the time, when two sets are unequal, Schwartz's heuristic will rapidly determine that they are unequal. (The rest of the time, it will be unable to decide if they are equal or not.) We improve the scheme so that given any two unequal sets, there is a high probability that the algorithm will determine they are unequal. The probability may be made so high that the lack of showing they are unequal is a good indication that they are equal.

We view this algorithm as a tool to be used for other applications, rather than because we are directly interested in testing set equality. Thus, it is important that we can prove that the probability of making a mistake is not dependent on the particular characteristics of the input string, otherwise our application could only use sequences of requests which had those characteristics. We accomplish this by constructing a class of algorithms and showing that for the worst input the probability that a randomly chosen algorithm will produce an error is low.

We will first present a simple, although incomplete algorithm. Let G be a group with operation $\oplus$ and denote the inverse of x in G by x^{-1}. Hash is a function chosen from a strongly universal$_\omega$ set which maps elements of the sets being constructed into G. One might implement this scheme with G being the set of bit strings of a certain length, and $\oplus$ being exclusive-or. This operation is easy to perform, and the inverse operation - the identity function - is even easier.

```
Given Sets $S_j$ for $j = 1$ to n
                  and elements $x_i$ for $i = 1$ to m,
  Set $V_i$ to a group element y, where y is a constant
  For each add($x_i$,$s_j$) request,
      set $V_j := V_j \oplus \text{hash}(x_i)$;
  For each delete($x_i$,$S_j$) request,
      set $V_j := V_j \oplus \text{hash}(x_i)^{-1}$;
  For each test($S_i$,$S_j$) request,
      output 'yes' if $V_i = V_j$, otherwise 'no';
```

There are two imaginable errors: our algorithm might say two equal sets were unequal, or it might say two unequal sets were equal. If $V_i \neq V_j$ then sets S_j and S_i must be unequal. Thus, we cannot say two equal sets are unequal. We will now bound the probability of saying two unequal sets are equal.

If sets S_j and S_i are unequal then either set S_j or set S_i must have an element not contained in the other. We enumerate the elements of set S_i by $S_{i,k}$. Without loss of generality we will assume S_i has an element, $S_{i,1}$ not contained in S_j. If $V_i = V_j$ then

$$\text{hash}(S_{i,1}) \oplus \text{hash}(S_{i,2}) \ldots = \text{hash}(S_{j,1}) \oplus \text{hash}(S_{j,2}) \ldots$$

which is true if and only if

$$\text{hash}(S_{i,1}) = \text{hash}(S_{i,2})^{-1} \oplus \text{hash}(S_{i,3})^{-1} \ldots \oplus \text{hash}(S_{j,1}) \oplus \text{hash}(S_{j,2}) \ldots$$

It follows from the definition of strongly universal$_\omega$ that for any bit string b, the probability that $\text{hash}(S_{i,1}) = b$ is independent of the value of b and the values of hash on other elements, so the probability that $\text{hash}(S_{i,1}) = \text{hash}(S_{i,2})^{-1} \oplus \text{hash}(S_{i,3})^{-1} \ldots \oplus \text{hash}(S_{j,1}) \oplus \text{hash}(S_{j,2}) \ldots$. is the reciprocal of the number of possible bit strings. Thus, the probability that $V_i = V_j$ when $S_i \neq S_j$ is the reciprocal of the number of possible bit strings. As the length of the bit strings produced by hash is increased, the probability of a mistake is decreased.

Notice that one indeed needs to use a strongly universal$_\omega$ set of functions, since the above reasoning requires that the value of $\text{hash}(S_1)$ be independent of the value of hash on all the other elements of sets S_i and S_j.

6. A Strongly Universal$_\omega$ Set

We can get a strongly universal$_\omega$ set of functions as follows. The techniques of [CW] give us the ability to use an associative memory which requires constant expected time per request. We assume the ability to generate random numbers. We will use these two abilities to create a partial function f defined only on all the inputs we have seen. The algorithm f uses is: if there is a value associated with its argument, x, in the associative memory, f(x) is that value. Otherwise, f(x) equals a randomly chosen value, and the algorithm stores x and that value in the associative memory. Thus, the value of hash on any element is independent of its values on any other elements.

The above argument suffices to prove:
Theorem: the probability of any equality test resulting in an incorrect answer, in the above algorithm, is exactly equal to the reciprocal of the number of elements in the group.

If the group elements are all bit string of length 100, the error probability per test would be $\frac{1}{2^{100}}$. It could be argued that this is less than the probability of a machine error in the extra time necessary to do a complete check.

We briefly mention some extensions to this technique. If we want to be able to test membership quickly as well as the other three requests, we can add a hash table with each set. If we want to form a set, S_i which is the symmetric difference of two other sets, S_j, S_k, then the value of V_i will be equal to V_j exclusive-ored with V_k and the group operation will be exclusive-or. Since V_i can be formed quickly, forming of symmetric difference can be added to the above three. However, finding the hash table that corresponds to a set which is the symmetric difference of two sets is not a fast operation so forming symmetric difference and testing membership cannot be combined. If in the above discussion exclusive-or is replaced by addition, a request for forming a set by multi-set union becomes possible in the same way forming symmetric difference was. We leave as exercises to the reader the proofs that it is improbable that mistakes will be made.

We now give two applications of this testing of set equality. A graph may be represented as a set of nodes and a set of edges. Testing labeled graphs for equality is now easy. Secondly the memory state of a computer may be represented as a set of pairs, each pair consisting of an address and the value stored in that address [Brown]. If a value in memory is changed, we delete the pair consisting of the address and the old value, and add the pair of the address and the new value. We can use a hash table to see if a memory state has been seen before, and thus whether the program is looping (see Cocke).

It is conceivable that the equality test could be extended to other requests. If the equality test could be extended adequately, a good part of a language like SETL might fit in such a scheme. However, a recent result by Yao suggests that it is impossible to find a fast test for two sets being disjoint [Yao].

References

[Adleman] Adleman, L. Private communication, Dec. 1977.

[Brassard] Brassard, G. Private communication, May 1977.

[Brown] Brown, D., Ph.D Dissertation, MIT.

[Cocke] Cocke, J., as told to P.C. Goldberg, "Partial Execution", in preparation.

[CGFMW] Carter, J.L., Gill, J., Floyd, R., Markowsky, G., and Wegman, M., "Exact and Approximate Membership Testers", Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, May 1978, pp.59-65.

[CW] Carter, J. L. & Wegman, M. N. "Universal Classes of Hash Functions," Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, May, 1977, pp.106-112 (newer version IBM RC-6687).

[Diffie] Diffie, W. and Hellman, M. "New Directions in Cryptography", IEEE Transactions on Information Theory (November 1976)

[Fak] Fak, V. "Repeated Use of Codes which Detect Deception" IEEE Transactions on Information Theory Vol. 25 No. 2 pp. 233-234

[FNPS] R. Fagin, J. Nievergelt, N. Pippenger, H. R. Strong "Extendible Hashing - A Fast Access Method for Dynamic Files", to appear in ACM Transactions on Database Systems.

[GMS] Gilbert, E.N., MacWilliams, F.J. and Sloane N.J.A. "Codes Which Detect Deception", The Bell System Technical Journal pp.405-424 (March 1947).

[Rivest] Rivest R., Shamir A., and Adleman L. "On Digital Signatures and Public-key Cryptosystems" MIT/LCS/TM-82.

[Schwartz] Schwartz J. T. SETL Newsletter.

[Yao] Yao, A., "Some Complexity Questions Related to Distributive Computing", Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing, May 1979, pp.209-213.