One-Way Accumulators: A Decentralized Alternative to Digital Signatures (Extended Abstract)

Josh Benaloh[1] and Michael de Mare[2]

[1] Clarkson University
[2] Giordano Automation

Abstract. This paper describes a simple candidate one-way hash function which satisfies a *quasi-commutative* property that allows it to be used as an accumulator. This property allows protocols to be developed in which the need for a trusted central authority can be eliminated. Space-efficient distributed protocols are given for document time stamping and for membership testing, and many other applications are possible.

1 Introduction

One-way hash functions are generally defined as functions of a single argument which (in a "difficult to invert" fashion) reduce their arguments to a predetermined size. We view hash functions, somewhat differently here, as functions which take two arguments from comparably sized domains and produce a result of similar size. In other words, a hash function is a function h with the property that $h: A \times B \rightarrow C$ where $|A| \approx |B| \approx |C|$. There is, of course, no substantial difference between this view and the traditional view except that this view allows us to define a special *quasi-commutative* property which, as it turns out, has several applications.

The desired property is obtained by considering functions $h: X \times Y \rightarrow X$ and asserting that for all $x \in X$ and for all $y_1, y_2 \in Y$,

$$h(h(x, y_1), y_2) = h(h(x, y_2), y_1).$$

This property is not at all unusual. Addition and multiplication modulo n both have this property as does exponentiation modulo n when written as $e_n(x, y) = x^y \bmod n$. Of these, only exponentiation modulo n has the additional property that (under suitable conditions), the function is believed to be difficult to invert.

This paper will describe how to use the combination of these two properties (quasi-commutativity and one-wayness) to develop a *one-way accumulator* which (among other applications) can be used to provide space-efficient cryptographic protocols for time stamping and membership testing.

2 Definitions

We begin by formalizing the necessary definitions.

Definition 1. A family of *one-way hash functions* is an infinite set of functions $h_\ell\colon X_\ell \times Y_\ell \to Z_\ell$ having the following properties:

1. There exists a polynomial P such that for each integer ℓ, $h_\ell(x,y)$ is computable in time $P(\ell, |x|, |y|)$ for all $x \in X_\ell$ and all $y \in Y_\ell$.
2. There is no polynomial P such that there exists a probabilistic polynomial time algorithm which, for all sufficiently large ℓ, will when given ℓ, a pair $(x,y) \in X_\ell \times Y_\ell$, and a $y' \in Y_\ell$, find an $x' \in X_\ell$ such that $h_\ell(x,y) = h_\ell(x',y')$ with probability greater than $1/P(\ell)$ when (x,y) is chosen uniformly among all elements of $X_\ell \times Y_\ell$ and y' is chosen uniformly from Y_ℓ.

Note that the above definition only requires that "collisions" of the form $h(x,y) = h(x',y')$ for given x, y, and y' are hard to find. That is, given x, y, y', it is, in general, hard to find a *preimage* x' such that $h(x,y) = h(x',y')$. It may in fact be easy, given $(x,y) \in X \times Y$, to find a pair $(x',y') \in X \times Y$ such that $h(x,y) = h(x',y')$. It must, however, be the case that for a given (x,y) pair, there are relatively few $y' \in Y$ for which an $x' \in X$ can practically be found such that $h(x,y) = h(x',y')$.

Note also that this definition does not require that the "hash" value be smaller than its arguments. However, the hash functions considered here will have the property that $|X| \approx |Y| \approx |Z|$.

It follows from the above definition that a family of one-way hash functions is itself a family of one-way functions. Work by Naor and Yung ([NaYu89]) and by Rompel ([Romp90]) has shown that one-way hash functions exists if and only if one-way functions exist which, in turn, exist if and only if secure signature schemes exist. It has also been shown ([ILL89]) that the existence of one-way functions is equivalent to the existence of secure pseudo-random number-generators.

Definition 2. A function $f\colon X \times Y \to X$ is said to be *quasi-commutative* if for all $x \in X$ and for all $y_1, y_2 \in Y$,

$$f(f(x,y_1),y_2) = f(f(x,y_2),y_1).$$

By considering one-way hash functions for which the range is equal to the first argument of the domain, i.e. $h\colon X \times Y \to X$, we can exploit the properties of one-way hash functions which also have the quasi-commutative property.

Definition 3. A family of *one-way accumulators* is a family of one-way hash functions each of which is quasi-commutative.

3 Motivation

The quasi-commutative property of one-way accumulators h ensures that if one starts with an initial value $x \in X$, and a set of values $y_1, y_2, \ldots, y_m \in Y$, then the *accumulated hash*

$$z = h(h(h(\cdots h(h(h(x, y_1), y_2), y_3), \cdots, y_{m-2}), y_{m-1}), y_m)$$

would be unchanged if the order of the y_i were permuted.

In addition, the fact that h is a one-way hash function means that given $x \in X$ and $y \in Y$ it is difficult to, for a given $y' \in Y$, find an $x' \in X$ such that $h(x, y) = h(x', y')$.[3]

Thus, if the values $y_1, y_2, \ldots, y_m$ are associated with users of a cryptosystem, the accumulated hash z of all of the y_i can be computed. A user holding a particular y_j can compute a partial accumulated hash z_j of all y_i with $i \neq j$. The holder of y_j can then (presumably at a later time) demonstrate that y_j was a part of the original hash by presenting z_j and y_j such that $z = h(z_j, y_j)$. A user who wishes to *forge* a particular y' would be faced with the problem of constructing an x' with the property that $z = h(x', y')$.

This approach does *not* enable users to hide their individual y_j since all of the y_j are necessary to compute the accumulated hash z (although the y_j may themselves be encryptions of hidden information). However, using a one-way accumulator in this way keeps each user from having to *remember* all of the y_j.

A general application of this basic trick is as an alternative to digital signatures for credential authentication: if all parties retain the result z of the accumulated hash, then at a later time, any party can present its (y_j, z_j) pair to any other party who can then compute and verify $h(y_j, z_j) = z$ to authenticate y_j.

It might, of course, be possible for a dishonest user to construct a false pair (x', y') such that $h(x', y') = z$ by combining the various y_i in one way or another. It will, however, be seen in section 5.1 that this is not practical. Other methods of computing false (x', y') pairs may also be possible. However, by restricting the choice of y', constructing "useful" false pairs can be made impractical.

It should be emphasized that the advantage of this approach over the naive "save everything you see" approach is simply one of storage. In terms of storage, this protocol is comparable to that of retaining a public-key for a central authority and using it to verify that y_j has been signed by the central authority. However, using the one-way accumulator method can obviate the need for a central authority altogether.

Two applications of one-way accumulators will be presented in section 5. The first is a method to construct a time stamping protocol in which participants can archive and time stamp their documents in such a way as to allow the time stamped documents to be revealed to others at a later time. A second

[3] The assertion that the composition formed by applying h many times is one-way is not strictly the same as asserting that h itself is one-way. This will be addressed in section 4.

application shows how a membership testing system can be constructed without having to maintain membership lists. In both applications, storage requirements are minimized without having to rely upon a (potentially corruptible) central authority.

4 Modular Exponentiation

For any n, the function $e_n(x, y) = x^y \bmod n$ is clearly quasi-commutative. The commonly used RSA assumption ([RSA78]) is that for "appropriately chosen" n, computing x from $e_n(x, y)$, y, and, n cannot be done in time polynomial in $|n|$ except in an exponentially small number of cases. In [Sham81], Shamir gives a proof which, when applied in this context, shows that for these appropriately chosen n, if root finding modulo n is hard, then the family e_n constitutes a family of one-way hash functions. However, even this may not be enough if the e_n are to be used as one-way accumulators. The reason for this is that repeated application of an e_n may reduce the size of the image so much that finding collisions becomes feasible.

To alleviate this problem, we restrict our n even further than do most.

Definition 4. Define a prime p to be *safe* if $p = 2p' + 1$ where p' is an odd prime.

Definition 5. We define n to be a *rigid integer* if $n = pq$ where p and q are distinct safe primes such that $|p| = |q|$.

It is not hard to see that for $n = pq$ to be a rigid integer larger than 100, each of p, q, $\frac{(p-1)}{2}$ and $\frac{(q-1)}{2}$ must be primes congruent to 5 modulo 6.

4.1 Composition

The advantage of using a rigid integer $n = pq$ is that the group of squares (quadratic residues) modulo n that are relatively prime to n has the property that it has size $n' = \frac{(p-1)}{2}\frac{(q-1)}{2}$ and the function $e_{n,y}(x) = x^y \bmod n$ is a permutation of this group whenever y and n' are relatively prime. Thus, if the factorization of n is hidden, "random" exponentiations of an element of this group are extremely unlikely to produce elements of any proper subgroup. This means that repeated applications of $e_n(x, y)$ are extremely unlikely to reduce the size of the domain or produce random collisions.

Although constructing rigid integers is somewhat harder than constructing ordinary "difficult to factor" integers, it is still quite feasible. The process would be to select "random" p' congruent to 5 modulo 6 until one is found such that p' and $2p' + 1$ are both prime. Approximately one out of every $(\ln p')^2$ of the p' selected will have this property. Once a suitable p' has been found, a suitable q' is selected similarly. This allows $n = pq = (2p' + 1)(2q' + 1)$ to be formed within approximately $2(\ln p')^2$ trials. Thus, if the modulus n is to be approximately 200 digits in length, approximately 10,000 candidates for each of p' and q' would be expected to be examined before suitable choices are found. This would mean executing roughly 20,000 primality tests on 100 digit integers – an amount of work which is not terribly unreasonable.

In some sense, rigid integers may be the hardest of all integers to factor. Most cryptographic applications which depend upon the difficulty of factoring suggest that n be chosen as a product of two comparably sized primes p and q and further suggest that $p-1$ and $q-1$ each contain large prime factors. Such n are suitable for our purposes also. However, taking these parameters to the extreme case in which each of $p-1$ and $q-1$ have the largest of possible prime factors (namely $(p-1)/2$ and $(q-1)/2$) provides additional beneficial properties which can be exploited by our applications.

4.2 Collisions

The one-way property of one-way accumulators rests not on the difficulty of finding arbitrary collisions, but rather upon the difficulty of finding collisions (or alternate preimages) with specific constraints.

If an accumulated hash z, is formed from a given set of values taken modulo n, a new item y can be *forged* by finding an x such that $z = x^y \bmod n$. If y is itself the result of a one-way hash, a prospective forger must, for a y that it can change but not select, compute a y^{th} root of z modulo n.

This, on the face of it, appears to be as hard as computing roots modulo a composite n which is believed to be computationally infeasible for large n of unknown factorization.

There are, however, other factors which may make the task easier for the prospective forger. First, together with z, the forger is provided with a number of roots of z modulo n. (These other roots are provided by the values used to form z.) Shamir, however, has shown ([Sham81]) that if basic root computation is difficult, then the roots $z^{1/r_1}, z^{1/r_2}, \ldots, z^{1/r_k}$ are insufficient to compute the value of $z^{1/\rho}$ unless ρ is a divisor of $R = \prod_{i=1}^{k} r_i$. Second, the forger may have had an opportunity to select some of the constituent y out of which the accumulated hash z was constructed. It is conceivable that a forger may weaken the system by choosing appropriate constituents which will facilitate a subsequent forgery.

We sketch below the result which says that even an actively participating (dynamic) forger cannot exist unless root finding is computationally feasible.

Theorem 6. *Suppose there exists a polynomial time algorithm $\mathcal{A}$ which when given x and n and a polynomial number of roots $y_1, y_2, \ldots, y_k$ and pre-selected indices $r_1, r_2, \ldots, r_k$ of x such that each $y_i^{r_i} \bmod n = x$ finds, for a given r, a y such that $y^r \bmod n = x$. Then there exists a polynomial time algorithm $\mathcal{B}$ which when given x, n, and $\rho = r/\gcd(r, r_1 r_2 \cdots r_k)$ will produce a y such that $y^\rho \bmod n = x$.* (In other words, the computation can be duplicated without the use of the roots $y_1, y_2, \ldots, y_k$.)

Proof. (sketch)

Algorithm $\mathcal{B}$ can be constructed from algorithm $\mathcal{A}$ as follows. Given x, n, and ρ, $\mathcal{B}$ computes $\hat{x} = x^{r_1 r_2 \cdots r_k} \bmod n$ and asks $\mathcal{A}$ for an r^{th} root of $\hat{x}$ modulo n by providing $\mathcal{A}$ with the appropriate roots of $\hat{x}$ which can be easily computed from x and the r_i. $\mathcal{A}$ will return a $\hat{y}$ such that $\hat{y}^r \bmod n = \hat{x}$. Let $g = \gcd(r, r_1 r_2 \cdots r_k)$. The quotients $\frac{r}{g}$ and $\frac{r_1 r_2 \cdots r_k}{g}$ are now relatively prime, and the extended Euclidean algorithm can be used to construct cofactors a and b such that

$$a\left(\frac{r}{g}\right) + b\left(\frac{r_1 r_2 \cdots r_k}{g}\right) = 1.$$

The desired root $x^{1/\rho} \bmod n$ can now be constructed as $x^{1/\rho} \bmod n = x^a \hat{y}^b \bmod n$ since $(x^a \hat{y}^b)^\rho \bmod n = x^{(ar/g)} \hat{y}^{(br/g)} \bmod n = x^{(ar/g)} x^{(br_1 r_2 \cdots r_k/g)} \bmod n = x \bmod n$. □

In short, this theorem shows that (unless general root finding is feasible) an r^{th} root of a given z modulo n can be computed only if one is given a set of known roots and indices $\{(x_i, r_i) : x_i^{r_i} \bmod n = z\}$ such that r is a divisor of $\prod r_i$.

It may, however, be possible for a forger to obtain a set of roots such that the product R of their indices *is* a multiple of the desired root index. But, it can be shown that the number of known roots which would have to be provided in order to have a non-negligible probability of their product being a multiple of a random number r selected later would be prohibitively large (see [KnTr76]). Asymptotically, for any polynomial P, it is the case $P(|n|)$ items can be combined into a single accumulated hash value with extremely high security. Numerically, even in a worst-case scenario in which an adversary is allowed to select all hash values (root indices) in advance, a 220 digit n would comfortably allow about 20 million items to be hashed with probability of forgery well below 10^{-30}. (See [Brui51], [Mitc68], and [LuWa69].)

A full asymptotic and numerical analysis will be included in the full version of this paper.

5 Applications

Two applications are described in this section.

5.1 Time Stamping

Haber and Stornetta ([HaSt90]) describe how documents can be *time stamped* by cryptographically chaining documents. By following the links in the chain, one can later determine where in the sequence a document occurred and thereby determine the relative positions of any two documents. This process, however, is somewhat cumbersome since it requires the active cooperation of other participants who have documents in the chain. Each link of the chain must be individually reconstructed to relocate the position of a document.

In the same work, Haber and Stornetta also describe a system by which documents are transmitted to a subset of the participants. The specific subset is determined by the document itself. With the appropriate cooperation of these participants, one can later substantiate to others that the document was sent at the claimed time.

Benaloh and de Mare ([BeMa91]) describe another method using a somewhat different model. They break time into rounds and assume the existence of broadcast channels (which can be simulated with any of a variety of consensus protocols — see, for example, [CGMA85], [Fisc83], [BenO83], and [Rabi83]). Benaloh and de Mare describe how time stamping can be accomplished without assumptions of cooperation. Within their model, they show how the amount of information which must be saved in each round of the protocol can be made proportional to the logarithm of the number of participants in the protocol. They pose as an open problem the question of whether the amount of information which must be saved can be made independent of the number of participants.

The time stamping protocol given here essentially solves the question posed by Benaloh and de Mare. Using modular exponentiation as a one-way accumulator, a simple protocol can be devised.

A Time Stamping Protocol. Before beginning, a rigid integer n is agreed to by all parties. This n can be supplied by a (trusted) outside source, constructed by a special purpose physical device, or (perhaps more likely) chosen by joint evaluation of a circuit for computing such an n which is supplied with random inputs by the participants (see [GMW86], [GMW87], [BGW88], [CCD88], [RaBe89], [Beav89], [BeGo89], [GoLe90], [MiRa90], and [Beav91] for work on secure multiparty computation). Since this n need be selected only once and may thereafter be used continuously, any extraordinary effort which may be required to construct such an n may be warranted.

Once n has been selected, a starting value x is agreed upon. This x may, for instance, be a representation of the current date. From this x, the value $x_0 = x^2 \bmod n$ is formed.

Each of the m participants takes any document(s) that it wishes to stamp in a given round and applies an agreed upon conventional one-way hash function to its document(s) to produce a y such that $y < n$. Let $y_1, y_2, \ldots, y_m$ denote the set of (conventionally hashed) documents to be stamped in a given round. Let $Y = \prod_{i=1}^{m} y_i$, and for each j let y'_j denote the product Y/y_j. The *time print* of the round z is computed as the accumulated hash

$$z = x_0^Y \bmod n = ((\cdots((x_0^{y_1} \bmod n)^{y_2} \bmod n)\cdots)^{y_m}) \bmod n.$$

The j^{th} participant also computes and maintains the partial accumulated hash

$$z_j = x_0^{y'_j} \bmod n$$

which is also easily computed.

Now, for the j^{th} participant to demonstrate at a later time that a given document (which presumably only it saved) has a claimed time stamp, the participant need only produce y_j and z_j. Anyone can check that $z_j^{y_j} \bmod n$ is equal to the time print z of the round and must therefore accept the time stamp of the document as legitimate. The claimant can then show that when the conventional hash function is applied to its document the value y_j is produced.

Is Forgery Possible? Before discussing whether or not forgery is possible, we must define precisely what forgery means within this context. A participant has the ability to time stamp many documents per round. These documents might contain contradictory information or promises. There is nothing, for instance, to stop a participant from time stamping a large number of predictions about the world series outcome and then (after the outcome is decided) revealing only the one time stamped document which correctly predicted the outcome.

Depending on the method of implementation, it might even be possible for a user who wishes to stamp (hashed) document y to, for example, submit (hashed) documents u and v for stamping where $y = uv$ and then later construct a time stamp for y out of the time stamps for u and v. Although this simple ploy can be remedied by requiring the submission of both pre-hash and post-hash documents (note that the documents may, of course, also be encrypted before any hashing to protect their contents), other similar ploys may be possible if the user knows the document for which a stamp is desired at the time of the stamp. This, however, does not pose a concern since we allow participants to stamp any and all documents within any round.

The only claim which we can make about forgery is that a user cannot produce a valid time stamp for a document that was not anticipated at the time indicated by the stamp. For example, an industrial spy who reads a patent application with a given date will not be able to change the name on the application and forge a time stamp to indicate an earlier date.

The results of theorem 6, however, show that forging unanticipated documents is infeasible.

5.2 Membership Testing

Suppose a large group of people (perhaps the attendees of a cryptography conference) want to develop a mechanism which will allow participants to recognize each other at a later time. Several solutions are possible.

The attendees could simply produce a membership list and distribute the list amongst themselves. However, this requires each member to maintain a large and bulky membership list. In addition, if the members do not want outsiders to know their identities, these membership lists would have to be carefully guarded by all members. Thus, it is never possible for a member to be identified to a non-member.

An alternative solution would be for the group to appoint a trusted secretary. The secretary can digitally sign "id cards" for each member and post its own public verification key. Each member need only remember its own signed information and the secretary's public key. At a later time, one member can be identified to another by providing its own signed id card. Additionally, it is possible to give the secretary's public key to outsiders so the members can identify themselves to non-members. The problem, of course, is that the secretary must be trusted to not produce additional "phony" id cards for non-members.

One-way accumulators offer a solution with the advantages of a single trusted secretary but without the need for such an authority. Each member selects a y_j consisting of its name and any other desired identifying characteristics. A base x is agreed upon, and the members exchange their information and compute the accumulated hash value

$$z = h(h(h(\cdots h(h(h(x, y_1), y_2), y_3), \cdots, y_{m-2}), y_{m-1}), y_m).$$

Each member saves the hash function h, its own y_j, and the value z_j which represents the accumulated hash of all y_i with $i \neq j$. For the holder of y_j to prove that it is a member of the group, it need only present the pair (y_j, z_j). By verifying that $h(y_j, z_j) = z$, any other participant can authenticate the membership of the holder of y_j. Note that it is not even necessary for each participant to retain the accumulated hash value z since each participant would hold its own (y_j, z_j) pair from which $z = h(y_j, z_j)$ can be easily generated.

Also, non-members can be given the hash function h and the value of the accumulated hash z. Thus, any member that wishes to can identify itself to a non-member without revealing the entire membership list.

In [Merk80], Merkle describes a similar application in which a directory of public keys is to be jointly maintained. He describes a "tree authentication" solution to the problem in which each user must retain its own key, a hash function h, and a number of additional partial hashes which is logarithmic in the number of participants. By using one-way accumulators, the same properties can be achieved while reducing to a constant the number of values which must be retained by each participant.

6 Other Applications, Generalizations, and Further Work

The idea of one-way accumulators can be applied to a variety of other problems. The special advantage offered by accumulators over signatures is that no one individual need know how to authenticate/sign/stamp a document or message. Thus, a class of applications of one-way accumulators is as a simple and effective method of forming collective signatures. There seem to be a variety of cryptographic problems which are closely related to membership testing, and it seems likely that such problems may be amenable to the approach of one-way accumulators. Many other applications may also be possible.

David Naccache has observed that the function $e_{n,c} = x^y c^{y-1} \bmod n$ is quasi-commutative for all constants c. This is a direct generalization of the function $e_n(x, y) = x^y \bmod n = e_{n,1}(x, y)$ used within this paper. A possible advantage of this more general form is that it facilitates the use of efficient Montgomery processors.[4] Naccache also observes that the Dixon polynomial generating function

[4] Montgomery processors can, for certain constants c which depend solely upon n, compute the modular product $abc \bmod n$ as quickly as ordinary processors can compute the integer product ab.

$$g(x,k) = \sum_{i=0}^{\lfloor k/2 \rfloor} \frac{k}{k-i} \binom{k-i}{i} x^{k-2i}$$

is quasi-commutative. It is not known whether these functions have appropriate one-way properties.

Clearly the existence of one-way accumulators implies the existence of one-way functions. The question of whether or not the existence of one-way accumulators is implied by the existence of arbitrary one-way functions is an area for future research. No relationship is known between the existence of one-way accumulators and that of one-way trap-door functions.

A related open question is that of whether a candidate one-way accumulator can be found which does *not* have a trap-door. There is no apparent reason why this should not be possible, and such a function could alleviate the need for the secure multiparty computation required to select an appropriate modulus n for the function $e_n(x,y) = x^y \bmod n$.

Acknowledgements

The authors would like to express their thanks to Narsim Banavara, Laurie Benaloh, Ernie Brickell, Joshua Glasser, David Greenberg, Kevin McCurley, David Naccache, Janice Searleman, Satish Thatte, Dwight Tuinstra, and anonymous reviewers for their helpful comments and suggestions regarding this work. The authors would also like to express their thanks to Paul Giordano.

References

[Beav91] **Beaver, D.** "Efficient Multiparty Protocols Using Circuit Randomization." *Advances in Cryptology — Crypto '91*, ed. by J. Feigenbaum in *Lecture Notes in Computer Science*, vol. 576, ed. by G. Goos and J. Hartmanis. Springer-Verlag, New York (1992), 420–432.

[Beav89] **Beaver, D.** "Multiparty Protocols Tolerating Half Faulty Processors." *Advances in Cryptology — Crypto '89*, ed. by G. Brassard in *Lecture Notes in Computer Science*, vol. 435, ed. by G. Goos and J. Hartmanis. Springer-Verlag, New York (1990), 560–572.

[BeGo89] **Beaver, D. and Goldwasser, S.** "Multiparty Computation with Faulty Majority." *Proc. 30th IEEE Symp. on Foundations of Computer Science*, Research Triangle Park, NC (Oct.–Nov. 1989), 468–473.

[BeMa91] **Benaloh, J. and de Mare, M.** "Efficient Broadcast Time-Stamping." *Clarkson University Department of Mathematics and Computer Science TR 91-1.* (Aug. 1991).

[BenO83] **Ben-Or, M.** "Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols." *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, Montreal, PQ (Aug. 1983), 27–30.

[BGW88] **Ben-Or, M., Goldwasser, S., and Wigderson, A.** "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation." *Proc. 20st ACM Symp. on Theory of Computation*, Chicago, IL (May 1988), 1–10.

[Brui51] **de Bruijn, N.** "The Asymptotic Behaviour of a Function Occurring in the Theory of Primes." *Journal of the Indian Mathematical Society 15.* (1951), 25–32.

[CCD88] **Chaum, D., Crépeau, C., and Damgård, I.** "Multiparty Unconditionally Secure Protocols." *Proc. 20st ACM Symp. on Theory of Computation*, Chicago, IL (May 1988), 11–19.

[CGMA85] **Chor, B., Goldwasser, S., Micali, S., and Awerbuch, B.** "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults." *Proc. 26th IEEE Symp. on Foundations of Computer Science*, Portland, OR (Oct. 1985), 383–395.

[Denn82] **Denning, D.** *Cryptography and Data Security*, Addison-Wesley, Reading, Massachusetts (1982).

[Fisc83] **Fischer, M.** "The Consensus Problem in Unreliable Distributed Systems", *Proc. 1983 International FCT-Conference*, Borgholm, Sweeden (Aug. 1983), 127–140. Published as *Foundations of Computation Theory*, ed. by M. Karpinski in *Lecture Notes in Computer Science*, vol. 158, ed. by G. Goos and J. Hartmanis. Springer-Verlag, New York (1983).

[GMW86] **Goldreich, O., Micali, S., and Wigderson, A** "Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design." *Proc. 27th IEEE Symp. on Foundations of Computer Science*, Toronto, ON (Oct. 1986), 174–187.

[GMW87] **Goldreich, O., Micali, S., and Wigderson, A** "How to Play Any Mental Game or A Completeness Theorem for Protocols with Honest Majority." *Proc. 19st ACM Symp. on Theory of Computation*, New York, NY (May 1987), 218–229.

[GoLe90] **Goldwasser, S. and Levin, L.** "Fair Computation of General Functions in Presence of Immoral Majority." *Advances in Cryptology — Crypto '90*, ed. by A. Menezes and S. Vanstone in *Lecture Notes in Computer Science*, vol. 537, ed. by G. Goos and J. Hartmanis. Springer-Verlag, New York (1991), 77–93.

[HaSt90] **Haber, S. and Stornetta, W.** "How to Time-Stamp a Digital Document." *Jounal of Cryptology 3.* (1991), 99–112.

[KnTr76] **Knuth, D. and Trabb Pardo, L.** "Analysis of a Simple Factorization Algorithm." *Theoretical Computer Science 3.* (1976), 321–348.

[ILL89] **Impagliazzo, R., Levin, L., and Luby, M.** "Pseudorandom Generation from One-Way Functions." *Proc. 21st ACM Symp. on Theory of Computation*, Seattle, WA (May 1989), 12–24.

[LuWa69] **van de Lune, J. and Wattel, E.** "On the Numerical Solution of a Differential-Difference Equation Arising in Analytic Number Theory." *Mathematics of Computation 23.* (1969), 417–421.

[Merk80] **Merkle, R.** "Protocols for Public Key Cryptosystems." *Proc. 1980 Symp. on Security and Privacy*, IEEE Computer Society (April 1980), 122–133.

[MiRa90] **Micali, T. and Rabin, T.** "Collective Coin Tossing Without Assumptions nor Broadcasting." *Advances in Cryptology — Crypto '90*, ed. by A. Menezes and S. Vanstone in *Lecture Notes in Computer Science*, vol. 537, ed. by G. Goos and J. Hartmanis. Springer-Verlag, New York (1991), 253–266.

[Mitc68] **Mitchell, W.** "An Evaluation of Golomb's Constant." *Mathematics of Computation 22.* (1968), 411–415.

[NaYu89] **Naor, M.** and **Yung, M.** "Universal One-Way Hash Functions and their Cryptographic Applications." *Proc.* 21st *ACM Symp. on Theory of Computation*, Seattle, WA (May 1989), 33–43.

[RaBe89] **Rabin, T.** and **Ben-Or, M.** "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority." *Proc.* 21st *ACM Symp. on Theory of Computation*, Seattle, WA (May 1989), 73–85.

[Rabi83] **Rabin, M.** "Randomized Byzantine Generals." *Proc.* 24th *IEEE Symp. on Foundations of Computer Science*, Tucson, AZ (Nov. 1983), 403–409.

[Romp90] **Rompel, J.** "One-Way Functions are Necessary and Sufficient for Secure Signatures." *Proc.* 22nd *ACM Symp. on Theory of Computation*, Baltimore, MD (May 1990).

[RSA78] **Rivest, R.**, **Shamir, A.**, and **Adleman, L.** "A Method for Obtaining Digital Signatures and Public-key Cryptosystems." *Comm. ACM 21*, 2 (Feb. 1978), 120–126.

[Sham81] **Shamir, A.** "On the Generation of Cryptographically Strong Pseudo-Random Sequences." *Proc. ICALP*, (1981).