Proofs of Work and Bread Pudding Protocols

Abstract

We formalize the notion of a *proof of work*. In many cryptographic protocols, a prover seeks to convince a verifier that she possesses knowledge of a secret or that a certain mathematical relation holds true. By contrast, in a proof of work, a prover demonstrates that she has performed a certain amount of computational work in a specified interval of time. Proofs of work have served as the basis of a number of security protocols in the literature, but have hitherto lacked careful characterization.

We also introduce the dependent idea of a *bread pudding protocol*. Bread pudding is a dish that originated with the purpose of re-using bread that has gone stale [10]. In the same spirit, we define a bread pudding protocol to be a proof of work such that the computational effort invested in the proof may also be harvested to achieve a separate, useful, and verifiably correct computation. As an example of a bread pudding protocol, we show how the MicroMint scheme of Rivest and Shamir can be broken up into a collection of proofs of work. These proofs of work can not only serve in their own right as mechanisms for security protocols, but can also be harvested in order to shift the burden of the MicroMint minting operation onto a large group of untrusted computational devices.

Keywords: distributed computation, proof of work, puzzle, hash function, MicroMint

1 Introduction

Proof protocols, whether interactive or non-interactive, serve as the cornerstone of almost every data security protocol. In a typical cryptographic

scenario, one party, the prover, aims to convince another party, the verifier, that it possesses a secret of a certain form, or that a certain mathematical statement holds true. For example, in the Schnorr identification protocol, the prover seeks to demonstrate possession of a secret key corresponding to a specific authenticated public key pair.

In this paper, we deviate from the standard cryptographic aim of proving knowledge of a secret or the truth of a mathematical statement. Instead, our goal is to formalize the notion of a *proof of work*. This is a protocol in which a prover demonstrates to a verifier that she has expended a certain level of computational effort in a specified interval of time. Although not defined as such or treated formally, proofs of work have been proposed as a mechanism for a number of security goals, including server access metering, construction of digital time capsules, and protection against spamming and other denial-of-service attacks [2, 3, 4, 5, 6, 9].

The contribution of this paper is twofold. First, we formalize the notion of a proof of work. We sketch definitions of a proof of work and of related concepts. As mentioned above, proofs of work have a demonstrated utility in a number of data security applications. A drawback to their use, however, is the fact that they impose a significant computational load in excess of that associated with most cryptographic protocols. This observation motivates the second contribution of our paper: the idea of *bread pudding protocols*. Bread pudding is a dish that originated with the purpose of re-using bread that has gone stale [10]. In the same spirit, we define a bread pudding protocol to be a proof of work such that the computational effort invested in the proof may also harvested to achieve a separate, useful, and verifably correct computation.

Ostrovsky [7] has proposed as an alternative to micropayment schemes the idea of having a client pay for access to a resource by offering a small amount of her computational power. A bread pudding protocol enables the fusion of this idea with the idea of a proof of work. The consequence is that the computational contribution of a client (i.e., the computational micropayment) can be quickly verified for correctness, and can simultaneously be used in security protocols. As an example of a bread pudding protocol, we consider the MicroMint scheme of Rivest and Shamir [8]. We show how the task of minting in this scheme can be partitioned into a collection of small proofs of work. These proofs of work can not only serve in their own right as mechanisms for security protocols, but can also be used to shift the burden of the MicroMint minting operation onto a large group of untrusted computational devices.

The remainder of the paper is organized as follows. In Section 2, we gives sketches of formal definitions for the idea of a proof of work and related concepts. We also define bread pudding protocols in this section. In Section 3, we offer a brief survey of previously proposed data security protocols employing proofs of work. We present our bread pudding protocol for the MicroMint minting operation in Section 4, and conclude in Section 5 with some ideas for future research.

2 Previous work

A number of security protocols in the literature have relied on the use of proofs of work. Researchers have not previously formalized the notion of a proof of work, however, and have adopted a wide ranging terminology to describe their constructions. In reviewing their work here, we use the term "puzzle". A puzzle is a special class of proof of work that we define in Section 3. It suffices here to think of a puzzle as a computational problem that is moderately hard, and whose difficulty is ensured by utilization of cryptographic primitives.

Dwork and Naor [2] were perhaps the first to advocate the use of puzzles as a way of attaching a computational cost to resource allocation requests. They propose the use of cryptographic puzzles as a deterrent to spam. They describe puzzle functions based on extraction of square roots over prime moduli; on the Fiat-Shamir signature scheme; and on the (broken) Ong-Schnorr-Shamir signature scheme.

Puzzles in the Dwork and Naor scheme are based on a hash of the time, destination, and the message. The sender of a piece of e-mail is required to enclose a solution to a given puzzle. Dwork and Naor also introduce the idea of a puzzle function with a trap door, i.e., a function that is moderately hard to compute without knowledge of the secret key, but easy to compute given this key. The availability of trap doors allows designated authorities to generate "postage" without significant expenditure of resources.

Gabber et al. [4] propose a method of controlling spam that may be regarded as an extension of [2], and also involves use of puzzles. Part of their proposal involves the use of a server that distributes permission – known as a handshake – for a sender to transmit mail to a recipient. This handshake is granted upon receipt of a valid puzzle solution from the sender.

Juels and Brainard [6] propose a similar use of puzzles as a deterrent against denial-of-service attacks against connection protocols such as SSL.

In their scheme, if a malicious party mounts an attack against a server by making many connection requests, the server begins to require clients to solve puzzles in order to initiate requests. Their scheme can be extended to non-attack scenarios in which equitable distribution of resources is desired.

Franklin and Malkhi [3] describe a scheme that makes use of puzzles for third-party verifiable usage metering. A Web site administrator requires users of her site to solve a puzzle for every access. To demonstrate to an auditor that her site has received a certain amount of usage, she presents the auditor with an audit log consisting of the set of solved puzzles. The auditor verifies the correctness of the audit log. The underlying assumption in this scheme is that many users will have a combined computational power far exceeding that readily available to the site administrator.

Goldschlag and Stubblebine [5] propose a scheme in which a puzzle serves as a mechanism to delay revelation of a secret for a desired period of time. Their aim is to enable verification of the fact that an, e.g., lottery has been properly administered at the same time that they ensure against premature disclosure of the associated secrets.

Rivest et al [9] discuss the creation of digital time capsules, employing a construction which they call a "time-lock puzzle". Their aim is to encrypt data in such a way that the decryption time can be carefully controlled. By discarding the encryption secrets, the data can thus be protected for a period of time designated by the creator. One important feature of the Rivest et al scheme is that the only feasible way to solve their proposed puzzle is sequential and deterministic. This is in contrast to most other puzzle constructions in the literature, where solutions may be sought using parallel computation.

3 Definitions

In this section, we offer formal definitions of the notion of a proof of work and of related concepts. These definitions must be regarded as sketches: we defer the exposition of thoroughly rigorous definitions to the full version of this paper. Let us begin by defining an *interactive proof of work*. This is a multi-round protocol executed by a prover P and a verifier V. The prover has a (bounded) memory of size m, and is permitted to perform an arbitrarily large amount of computation prior to the protocol execution. Both P and V have access to private coins, and may produce an arbitrarily large number of coin flips prior to the protocol execution. At the end of the

protocol, V decides either to accept or reject.

We define the *start time* t_s of the protocol to be the time at which the verifier initiates its first round of communication. The *completion time* t_c is the time at which the last round of the protocol is complete. The aim of a proof of work is to enable P to demonstrate that she has performed a certain amount of computation within the time interval $[t_s, t_c]$.

We say that a proof of work has a (w, p)-*workload* if for security parameter l and any integer u, there does not exist a prover P that is capable of convincing the verifier V to accept with probability greater than $p+O(m/l^u)$ while performing fewer than w steps of computation.

For many applications, an important requirement on proofs of work is that they be independent of one another. This is to say that work by the prover on one proof of work does not significantly diminish the hardness of a different proof of work. Let us suppose that a particular proof of work $\mathcal{PW}$ has a (w, p)-workload. Let $\mathcal{PW}_1, \mathcal{PW}_2, \ldots, \mathcal{PW}_k$ be instances of $\mathcal{PW}$ generated using independent verifier coin flips. Let $\hat{\mathcal{PW}}$ be a single proof of work interleaving all of these instances in some manner. We say that the proof of work $\mathcal{PW}$ has *independence* if the following is true. Let $X_1, X_2, \ldots, X_k$ be independent random variables such that the probability that $X_i = 1$ is $1 - p$ and the probability that $X_i = 0$ is p; let $X = \sum_i X_i$. Then for any formulation of $\hat{\mathcal{PW}}$, and any integer u, the combined proof of work $\hat{\mathcal{PW}}$ has workload $(wi, \mathsf{pr}[X = i] + O(1/l^u))$ for any integer i. In other words, the workload of interleaved proofs of work is not significantly less than the workload of proofs of work composed in a strictly serial fashion.

Let z be the maximum number of computational steps performed in a proof of work protocol by the verifier. We denote the *efficiency* of the proof of work by z/w. Thus, an efficient proof of work is one that involves relatively little computation on the part of the verifier. Efficiency is, of course, a highly desirable property in any proof of work.

We may also create a proof of work that is *non-interactive*. This is accomplished by simulating the behavior of the verifier V. Let c_V denote the private coin flips of V. In order to ensure that the protocol remains secure, it is necessary to generate c_V in a manner that cannot be effectively controlled by the prover. By analogy with non-interactive proofs for standard cryptographic properties, we accomplish this by reference to a public source of randomness or by some other appropriate means such as, e.g., generating c_V using the hash of some protocol-specific value. In this case, the start time t_s of the protocol is the time when the public source of randomness or the protocol-specific value is received by P.

In a two-round protocol, the verifier initiates the protocol by sending a query to the prover.[1] In this case, the information sent by the verifier constitutes a self-contained computational problem. We refer to this information as a *puzzle*, and information sent by the prover that causes acceptance by the verifier as a valid *puzzle solution*.

The final definition we present here is that of a *bread pudding protocol*. Let C be a large computational problem, such as, e.g., a large financial or scientific problem. Say that it is infeasible for any computational device to solve C in fewer than n computational steps. We say that $\mathcal{PW}$ is a bread pudding protocol for C if $\mathcal{PW}$ has independence and if an entity with access to the transcripts of proofs of work for some set of $k > 0$ independent instances $\mathcal{PW}_1, \mathcal{PW}_2, \ldots, \mathcal{PW}_k$ of $\mathcal{PW}$ can solve C in fewer than n computational steps.

3.1 Example of a proof of work

In order to make our definitions more concrete, we now present an example of a proof of work. This proof of work is very similar to that employed in several proposed security protocols, including those in [4, 6]. It also serves as the basis for our bread pudding protocol for MicroMint in Section 4. This proof of work requires two rounds.

Example 1 *Let h represent a one-way function. The verifier V generates a random bitstring x of length n and computes the image $y = h(x)$. Let x' be the first $l-k$ bits of x. V sends the pair (x', y) to P. In order to complete the proof of work successfully, P must calculate a valid pre-image $\tilde{x}$ of y. Observe that (x', y) constitutes a puzzle, while a valid $\tilde{x}$ constitutes a puzzle solution.*

The workload associated with this proof of work may be characterized by the following theorem. We prove this in the random oracle model for the one-way function h, treating an oracle query as a computational step.

Theorem 1 *The proof of work given in Example 1 has a (w, p)-workload for any integer $w \in [1, 2^l]$ and $p = w/2^l$.*

Proof: As we are considering h in the random oracle model, we may think of it as consisting of a random access tape in which cell i contains

[1] If the prover initiates the protocol, then the second round has no influence on the reply of the prover, so that the protocol can effectively be reduced to one round.

an independently generated random l-bit value $h(i)$. In order to present a pre-image of y in the proof of work without guessing, P must have in its memory the contents of a cell i such that $h(i) = y$. The probability that this is the case prior to the first round of the protocol is at most $m/2^l$. After w steps of work, the probability of the prover querying on a correct cell i is at most $w/2^l$. Therefore, the probability that P successfully completes the protocol without guessing is less than $(m + w)/2^l$. The probability of successful guessing on the part of the prover is 2^l. The result follows. ∎

4 A Bread Pudding Protocol for MicroMint

As an example of a bread pudding protocol, we consider the highly computationally intensive operation of minting in the MicroMint scheme. We show how to partition this task into a collection of proofs of work, enabling minting to be distributed among a collection of low power, untrusted entities. This is done without allowing the successful collisions (corresponding to micro-coins) to be "stolen" by the prover. Let us begin by describing how MicroMint works.

4.1 MicroMint

MicroMint is a micropayment system developed by Rivest and Shamir. Its security is based on the hardness of finding hash function collisions [8]. A coin in this scheme consists of a k-way hash function collision, that is, a set $\{x_1, x_2, \ldots, x_k\}$ of pre-images that map to a single image. By making coin values small, keeping records of spent coins, and embedding user identities in coins, it is possible to create strong deterrents to coin theft and double-spending.

The security of MicroMint against forgery derives from the large base computational costs associated with the minting operation. With appropriate parameterization of the scheme, minting a single coin is difficult, while the marginal cost associated with minting many coins is relatively small. (The use of k-way collisions, rather than 2-way collisions, increases the computational threshold required for producing the first coin.) Thus, minting requires a substantial base investment in hardware. For forgery to be successful, it must take place on too large a scale to make the effort worthwhile. By limiting the period of validity of a given coin issue and computing the issue over an extended period of time, the minter can even make the job of a forger harder than his own.

Suppose that the hash function h used for minting maps n-bit pre-images to n-bit images. The process of finding collisions may be thought of as that of throwing balls uniformly at random into a set of 2^n bins. Throwing a ball corresponds in this model to choosing a pre-image x and placing it in the bin with index $h(x)$. When k balls land in a single bin, they together constitute a coin.

If n is to be large enough to ensure an adequate level of security, the storage overhead associated with maintaining 2^n bins will be prohibitively large. Rivest and Shamir thus describe the following variation on the basic scheme. Let $n = t + u$. A ball (pre-image) x is considered valid only if the t least significant bits of $h(x)$ match some pre-selected, random value s. (If invalid, the ball may be considered to miss the set of bins.) A valid ball is thrown into one of a set of 2^u bins, according to the value of the u most significant bits. With the right parameterization, the computational effort associated with minting is still high, but the number of bins is smaller.

Note that to prevent a potential forger from initiating her effort prior to a given coin issue, it is possible in Rivest and Shamir's original scheme to key the hash function h with a secret value r that is only released on the issue date. For additional details and descriptions of a number of variants, the reader is advised to see [8].

4.2 Bread pudding with a little mint

We now demonstrate a simple bread pudding protocol for MicroMint, that is, a MicroMint variant in which the computation associated with minting may be embodied in a set of small puzzles. Let h be a suitable hash function and $\|$ denote string concatenation. We define a ball to be a triplet (i, x, y), where $y = h(r \parallel i)$ and r is a secret value as above. A valid ball is one in which the first t bits of $h(x \parallel y)$ are equal to s. The bin into which a ball is thrown is determined by the u most significant bits of $h(x \parallel y)$.

The computational cost associated with minting in this MicroMint variant remains the same as in the original scheme. Verifying a coin in the variant requires twice the number of hashes. The advantage of the variant scheme, however, is that the problem of finding a single, valid ball may be distributed as a small puzzle. By distributing enough of these puzzles, the minter may offload the majority of the computation associated with the minting operation.

In this scheme, the puzzle distributed to a client (prover) consists of the triple (s, t, y) for some correctly formulated value y. The task of the client is

to find a value x such that the first t bits of $h(x \parallel y)$ are equal to s, i.e., such that (i, x, y) is a valid ball. This puzzle requires an average computational effort of 2^{t-1} hashes. It may easily be seen to be a $(2^{t-1}, \frac{1}{2})$-workload proof of work, in accordance with the definitions in Section 3. Note that the secret value r is not included in a puzzle. Thus, even when minting is performed by way of puzzles, this secret value need only be released on the day of coin issue, so that the same security is achieved as in the original scheme.

Rivest and Shamir propose sample parameters in their paper of $k = 4$, $n = 52$, and $t = 21$ for achieving a viable minting setup. Thus, the puzzle based on finding a valid ball requires an average of 2^{20} hash operations. This is, as it happens, exactly the hardness of the puzzle proposed in [4], requiring about 2 seconds on a 266 MHz Pentium II processor under the hash function MD5. If the minter offloads the problem of finding valid balls onto clients, then his own computational effort is $k2^u = 2^{33}$ hash functions. This be computed in a few days on a standard desktop computer.

The 2^{33} puzzles of the hardness described here represent a great deal of computation to offload onto clients. For this reason, it may be desirable to make the puzzles somewhat easier. We can do this as follows, without changing the hardness of the minting operation or increasing the memory requirements of the minter. Let us require that y in a valid ball have v leading $'0'$ bits, and that only the first $t - v$ bits in $h(x \parallel y)$ be equal to a value s. Now a puzzle requires only 2^{t-v-1} hash computations on average, the reduced computational burden being shifted onto the minter in its fomulation of a puzzle.

5 Conclusion: Some Open Problems

We conclude by offering brief mention of some open problems motivated by this paper. The first of these is the problem of devising other bread pudding protocols. Other examples of bread pudding protocols would be desirable not only in themselves, but perhaps as a step toward defining a large class of computational problems amenable to parititioning into puzzles. Another open problem relates to proving results about the workloads of puzzles. We offer in this paper a proof in the random oracle model of the workload associated with a common puzzle type based on hash function inversion. Of use would be a stronger result more precisely characterizing the required properties of hash functions for this purpose. This line of exploration might yield additional results. For example, since proofs of

work generally involve only a few seconds of computation, it seems likely that weak cryptographic functions would serve in lieu of the conventional strong ones. This might yield more efficient proofs of work, and might have the interesting incidental consequence (as in [2]) of furnishing a use and haven for certain cryptographic algorithms that have been broken in a conventional sense.

References

[1] M. Blum and H. Wasserman. Software reliability via run-time result-checking. *Journal of the ACM.* To appear. Preliminary version: 'Program Result-Checking: A Theory of Testing Meets a Test of Theory,' Proc. 35th IEEE FOCS, 1994, pp. 382-392.

[2] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Proc. CRYPTO 92*, pages 139–147. Springer-Verlag, 1992. Lecture Notes in Computer Science No. 740.

[3] M.K. Franklin and D. Malkhi. Auditable metering with lightweight security. In R. Hirschfeld, editor, *Proc. Financial Cryptography 97 (FC 97)*, pages 151–160. Springer-Verlag, 1997. Lecture Notes in Computer Science No. 1318.

[4] E. Gabber, M. Jakobsson, Y. Matias, and A. Mayer. Curbing junk e-mail via secure classification. In R. Hirschfeld, editor, *Financial Cryptography '98.* Springer-Verlag, 1998.

[5] D. Goldschlag and S. Stubblebine. Publicly verifiable lotteries: Applications of delaying functions. In R. Hirschfeld, editor, *Financial Cryptography '98.* Springer-Verlag, 1998.

[6] A. Juels and J. Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *Symposium on Networks and Distributed Security Systems (NDSS '99)*, 1999. To appear.

[7] R. Ostrovsky. A proposal for internet *computational* commerce: How to tap the power of the WEB, 1998. Presentation at CRYPTO '98 Rump Session.

[8] R.L. Rivest and A. Shamir. PayWord and MicroMint–two simple micropayment schemes. *CryptoBytes*, 2(1):7–11, Spring 1996.

[9] R.L. Rivest, A. Shamir, and D. Wagner. Time-lock puzzles and timed-release crypto. To appear, 10 March 1996.

[10] Irma S. Rombauer and Marion Rombauer. Bread-pudding with meringue (six servings). In *Joy of Cooking*, page 751. Penguin Group, 1997.