(Short Paper) CommitCoin: Carbon Dating Commitments with Bitcoin*

Jeremy Clark[1] and Aleksander Essex[2]

[1] Carleton University
clark@scs.carleton.ca
[2] University of Waterloo
aessex@cs.uwaterloo.ca

Abstract. In the standard definition of a commitment scheme, the sender commits to a message and immediately sends the commitment to the recipient interested in it. However the sender may not always know at the time of commitment who will become interested in it. Further, when the interested party does emerge, it could be critical to establish when the commitment was made. Employing a proof of work protocol at commitment time will later allow anyone to "carbon date" when the commitment was made, approximately, without trusting any external parties. We present CommitCoin, an instantiation of this approach that harnesses the existing computational power of the Bitcoin peer-to-peer network; a network used to mint and trade digital cash.

1 Introductory Remarks

Consider the scenario where Alice makes a discovery. It is important to her that she receives recognition for her breakthrough, however she would also like to keep it a secret until she can establish a suitable infrastructure for monetizing it. By forgoing publication of her discovery, she risks Bob independently making the same discovery and publicizing it as his own.

Folklore suggests that Alice might mail herself a copy of her discovery and leave the letter sealed, with the postal service's timestamp intact, for a later resolution time. If Bob later claims the same discovery, the envelope can be produced and opened. In reality, this approach does not work as (among other shortcomings) most postal services are pleased to timestamp and deliver unsealed empty envelopes that can be retroactively stuffed with "discoveries."

In our approach, Alice will use a commitment scheme to put the discovery in a "digital envelope" which can be opened at some later time, but only by Alice. Alice can safely disclose the commitment value to anyone, but she does not know ahead of time that Bob will rediscover her breakthrough. Alice might attempt to reach Bob by broadcasting the commitment value to as many people as possible or she might have a trusted/distributed third party timestamp it, however she is neither guaranteed to reach Bob, nor choose a party that Bob will trust.

* Full version available: http://eprint.iacr.org/2011/677

Instead we show that Alice can produce a commitment and later convince Bob that the commitment was made at roughly the correct time, premised on the assumption that she does not have unusual computational power. We call this "carbon dating." We show a general approach to carbon dating using moderately hard puzzles and then propose a specific instantiation. CommitCoin harnesses the existing processing power of the Bitcoin network without trusting it, and is designed to leave the commitment value evident in the public Bitcoin transcript in a way that does not destroy currency. We use CommitCoin to augment the verifiability of a real-world election.

2 Preliminaries and Related Work

Commitment Schemes. Briefly, $\mathsf{Comm}(m, r)$ takes message m and randomness r and produces commitment c. $\mathsf{Open}(c, m, r)$ takes the commitment and purported message and returns accept iff c is a valid commitment to m. Commitments should be binding and hiding. Respectively, it should be hard to find any $\langle m_1, m_2, r\rangle$ where $m_1 \neq m_2$ such that $\mathsf{Open}(\mathsf{Comm}(m_1, r), m_2, r)$ accepts, and it should be hard to find any $\langle m, r\rangle$ given c such that $\mathsf{Open}(c, m, r)$ accepts.

Secure Time-Stamping. Secure time-stamping [18] is a protocol for preserving the chronological order of events. Generally, messages are inserted into a hash chain to ensure their relative temporal ordering is preserved under knowledge of any subsequent value in the chain. The chain is constructed by a distributed time-stamping service (TSS) and values are broadcast to interested participants. Messages are typically batched into a group, using a hash tree [4,3,7,27] or an accumulator [5], before insertion in the chain. Time-stamping is a mature field with standardization[3] and commercial implementations.

A secure timeline is a "tamper-evident, temporally-ordered, append-only sequence" of events [24]. If an event E_{t_i} occurs at time t_i, a secure timeline can only establish that it was inserted after $E_{t_{i-1}}$ was inserted and before $E_{t_{i+1}}$ was. To determine t_i by consulting the chain, one must either trust the TSS to vouch for the correct time, or, to partially decide, trust a recipient of a subsequent value in the chain to vouch for when that value was received (if at t_j, we can establish $t_i < t_j$). However should conflicting values emerge, implying different hash chains, there is no inherent way to resolve which chain is correct beyond consensus.

Non-Interactive Time-Stamping. An approach closely related to carbon dating is non-interactive time-stamping [25]. In such a scheme, stampers are not required to send any message at stamping time. The proposed scheme is in the bounded storage model. At each time interval, a long random bitstring is broadcast to all parties. Stampers store a subset that is functionally dependent on the message they are time-stamping. Verifiers also captured their own subset, called a sketch, at every time interval. This allows verification of the timestamp by anyone who

[3] ISO IEC 18014-3; IETF RFC 3161; ANSI ASC X9.95

is participating in the protocol, but not by a party external to the protocol. By contrast, our notion of carbon dating allows verification by anyone but is not necessarily non-interactive.

Proof of Work. The literature considers applications of moderately hard functions or puzzles that take a certain amount of computational resources to solve. These are variably called pricing [14], timing [15], delaying [17], or cost [16,2] functions; and time-lock [29,6,22] or client [20,1,12,32,33,13,31,10,30] puzzles. Proof of work is sometimes used as an umbrella term [19]. Among other applications, proof of work can be used to deter junk email [14,16] and denial of service attacks [20,12,2,32,33], construct time-release encryption and commitments [29,6], and mint coins in digital currencies [28,2,26].

We consider proof of work as three functions: $\langle \mathsf{Gen}, \mathsf{Solve}, \mathsf{Verify} \rangle$. The generate function $p = \mathsf{Gen}(d, r)$ takes difficulty parameter d and randomness r and generates puzzle p. The solve function $s = \mathsf{Solve}(p)$ generates solution s from p. Solve is a moderately hard function to compute, where d provides an expectation on the number of CPU instructions or memory accesses needed to evaluate Solve. Finally, verification $\mathsf{Verify}(p, s)$ accepts iff s is a correct solution to p.

Time-Stamping & Proof of Work. Bitcoin is a peer-to-peer digital currency that uses secure time-stamping to maintain a public transcript of every transaction [26]. However new events (groups of transactions) are appended to the hash chain only if they include the solution to a moderately hard puzzle generated non-interactively from the previous addition. Peers compete to solve each puzzle and the solver is awarded newly minted coins. A secure timeline with proof of work provides a mechanism to both limit the creation of new currency and to make it computationally difficult to change a past event and then catch up to the length of the original chain (peers accept the longest chain as canonical).

3 Commitments with Carbon Dating

A protocol for carbon dating commitments is provided in Protocol 1. It is a natural application of proof of work protocols but one that does not seem to have been specifically noted in the literature before.[4] Alice commits to a message m and instantiates a puzzle p based on the commitment value c that will take, on expectation, Δt units of time to solve. Alice begins solving p. Should a new party, Bob, become interested in when c was committed to, Alice will later produce the solution s. When given s, Bob concludes that p, and thus c, were created Δt time units before the present time. Since p will not take exactly Δt to solve, there is some variance in the implied instantiation time. We consider the case where Bob is only interested in whether the commitment was made well before a specific time of interest, which we call the pivot time.

If useful, a few extensions to Protocol 1 are possible. It should be apparent that carbon dating can be used for any type of sufficiently random message

[4] Concurrent to the review of this work, it is independently proposed and studied [23].

PROTOCOL 1 (Commitments with Carbon Dating)

Input: Alice has message m at t_1.

Output: Bob decides if m was known by Alice prior to pivot time t_2.

The protocol:

1. Pre-instantiation: At t_0, Alice commits to m with randomness r by computing $c = \mathsf{Comm}(m, r)$. She then generates puzzle based on c with difficulty d (such that the time to solve it is approximately Δt) by computing $p = \mathsf{Gen}(d, c)$. She outputs $\langle c, p \rangle$.
2. Instantiation: At t_1, Alice begins computing $s = \mathsf{Solve}(p)$.
3. Resolution: At $t_3 = t_1 + \Delta t$, Alice completes $s = \mathsf{Solve}(p)$ and outputs $\langle s, m, r \rangle$. Bob checks that both $\mathsf{Verify}(s, \mathsf{Gen}(d, c))$ and $\mathsf{Open}(c, m, r)$ accept. If so, Bob decides if $t_3 - \Delta t \overset{?}{\ll} t_2$

(*e.g.,* plaintexts, ciphertexts, signatures, *etc.*) by replacing c in $\mathsf{Gen}(d, c)$ with the message. Second, the commitment can be guaranteed to have been made *after* a given time by, *e.g.,* including recent financial data in the puzzle instantiation [11]. Finally, the resolution period can be extended by instantiating a new puzzle with the solution to the current puzzle (assuming the puzzles are entropy-preserving; see [17] for a definition of this property).[5]

3.1 Puzzle Properties

For carbon dating, we require the proof of work puzzle to have specific properties. Consider two representative proof of work puzzles from the literature (and recall c is the commitment value and d is a difficulty parameter). The first puzzle (P_{rs}), based on repeated squaring, is to compute $\mathsf{Solve}(d, c, N) = c^{2^d} \mod N$ where $N = q_1 q_2$ for *unknown* large primes q_1 and q_2, and $2^d \gg N$ [29,6,21]. The second puzzle (P_{h}), based on hash preimages, is to find an x such that $y = \mathcal{H}(c, x)$ has d leading zeros (where $\mathcal{H}$ is a cryptographic hash function)[6] [16,1,2,26]. We contrast the properties of P_{rs} and P_{h} with the properties of an ideal puzzle scheme for carbon dating (P_{cd}).

P_{cd} should be moderately hard given a sufficiently random c as a parameter. P_{rs} requires d modular multiplications and P_{h} requires 2^{d-1} hashes on average. Neither precomputation, amortizing the cost of solving many puzzles, or parallelization should be useful for solving P_{cd}. Parallelization is useful in solving P_{h}, while P_{rs} is by design inherently sequential. Verify in P_{cd} should be efficient for

[5] It may be preferable to solve a chain of short puzzles, rather than a single long puzzle, to allow (by the law of large numbers) the average solution time to converge and to reduce the amount of time Bob must wait for the solution.

[6] Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^m$. Then for $d \leq m$, find any x such that $y \in (\{0\}^d \| \{0,1\}^{m-d})$.

anyone. This is the case in $\mathsf{P_h}$ but not $\mathsf{P_{rs}}$, where efficient verification requires knowing the factorization of N,[7] making $\mathsf{P_{rs}}$ useful only when the puzzle creator and solver are *different* parties.[8] When surveying the literature, we found that like $\mathsf{P_{rs}}$ and $\mathsf{P_h}$, each type of puzzle is either parallelizable or only verifiable by the puzzle creator. Designing a non-interactive, non-parallelizable puzzle appears to be an open problem.

Finally, we require a few properties specific to our scheme. It should be hard to choose c such that the puzzle is not moderately hard. Given $s = \mathsf{Solve}(\mathsf{Gen}(d, c))$ and $s' = \mathsf{Solve}(\mathsf{Gen}(d, c'))$, it should be hard to find any pair of puzzles such that $s = s'$. Further, it should not be efficient to convert $\langle s, c\rangle$ into $\langle s', c'\rangle$.

3.2 Limitations

Aside from a good candidate for $\mathsf{P_{cd}}$, the primary limitation to Protocol 1 is that the implied instantiation time is fuzzy. Carbon dating is best when the ratio between instantiation-to-pivot and pivot-to-resolution is maximized but the timing of the pivot is often unknowable. Another limitation is that Alice could commit to many different messages but only claim one. This excludes carbon dating (and non-interactive time-stamping) from, *e.g.,* predicting election results or game outcomes. Generally, the scheme only works for accepting a committed message from an exponentially large set. A final limitation is that Alice must devote a CPU to solely solving the problem for a long period of time. We address this last limitation with $\mathsf{CommitCoin}$, and then latter provide an example where the first two limitations are not as applicable.

4 Carbon Dating with Bitcoin

Bitcoin is a peer-to-peer digital currency. A simplification of the scheme is as follows. Participants are identified by a public signing key. A transaction includes a sender, receiver, and amount to be transferred (units of bitcoins are denoted BTC), and it is digitally signed by the sender and broadcast to the network. Transactions are batched together (into a "block") and then appended to a hash chain ("block chain") by solving the $\mathsf{P_h}$ hash puzzle on the block ($d = 53$ bits currently). The first node to broadcast a solution is awarded newly minted coins (currently 50 BTC) plus any transaction fees (currently optional). At the time of writing, one large Bitcoin mining pool, *Deepbit*, reports being able to compute 2^{42} hashes/second, while the network solves a puzzle on average every 10 minutes.[9]

[7] The totient of N serves as a trapdoor: compute $\delta = 2^d \mod \phi(N)$ and then $s = c^\delta \mod N$.

[8] Alice could use $\mathsf{P_h}$ with the smallest unfactored N from the RSA challenges. Assuming continued interest in factoring these numbers, Alice's solution will eventually be verifiable. However she risks (a) it being factored before she solves the puzzle or (b) it never being factored at all. It also assumes non-collusion between Alice and RSA (assuming they know the factors).

[9] `http://deepbit.net`; `http://blockexplorer.com/q/interval`

PROTOCOL 2 (CommitCoin)

Input: Alice has message m, key pair $\langle sk, pk \rangle$ associated with a Bitcoin account. Without loss of generality the account has a balance of >2 BTC.

Output: The Bitcoin block chain visibly containing the commitment to m.

The protocol:

1. Pre-instantiation: At t_0, Alice does the following:
   (a) Alice commits to m with randomness r by computing $c = \mathsf{Comm}(m, r)$.
   (b) Alice generates new temporary key pair $\langle sk', pk' \rangle$ with $sk' = c$.
2. Instantiation: At t_1, Alice does the following:
   (a) Alice generates transaction $\tau_1 = \langle pk \to pk', 2 \rangle$ to send 2 BTC from pk to pk' and signs it with randomness ρ: $\sigma_1 = \mathsf{Sign}_{sk}(\tau_1, \rho)$. She outputs $\langle \tau_1, \sigma_1 \rangle$ to the Bitcoin network.
   (b) Alice generates transaction $\tau_2 = \langle pk' \to pk, 1 \rangle$ to send 1 BTC from pk' back to pk and signs it with randomness ρ': $\sigma_2 = \mathsf{Sign}_{sk'}(\tau_2, \rho')$. She outputs $\langle \tau_2, \sigma_2 \rangle$ to the Bitcoin network.
3. Tag & Open: At t_2, after τ_1 and τ_2 have been finalized, Alice generates transaction $\tau_3 = \langle pk' \to pk, 1 \rangle$ to send the remaining 1 BTC from pk' back to pk and signs it with *the same* randomness ρ': $\sigma_3 = \mathsf{Sign}_{sk'}(\tau_3, \rho')$. She outputs $\langle \tau_3, \sigma_3 \rangle$ to the Bitcoin network.
4. Extraction: At t_3, Bob can recover c by extracting sk' from σ_2 and σ_3.

Remark: For simplicity we do not consider transaction fees.

4.1 CommitCoin protocol

If Alice can put her commitment value into a Bitcoin transaction, it will be included in the chain of puzzles and the network will provide carbon dating without Alice having to perform the computation herself. Bob only has to trust that Alice cannot produce a fraudulent block chain, longer than the canonical one and in less time. This idea has been considered on the Bitcointalk message board[10]in the context of the distributed network vouching for the timestamp. Our observation is that even if you do not trust the timestamp or any node in the network, the proof of work itself can be used to carbon date the transaction (and thus commitment value).

In a Bitcoin transaction, Alice has control over several parameters including her private key(s), her public key(s), and the randomness used in the signature algorithm which, importantly, is ECDSA. If she sets the receiver's public key[11]to be her commitment value c and sends 1 BTC to it, the 1 BTC will be unrecoverable (akin to burning money). We consider this undesirable for two reasons: (a) it is financially wasteful for Alice and (b) it is not being a good citizen of the Bitcoin community.

[10] `http://goo.gl/fBNnA`
[11] Technically, it is a fingerprint of the public key.

By setting c equal to a private key or the signature randomness and following the protocol, c itself will never directly appear in the transcript. To get around this, Alice sets c to the private key of a new account and then purposely leaks the value of the private key by signing two different transactions with the same randomness. The CommitCoin protocol is given in Protocol 2. Since c is randomized, it has sufficient entropy to function (temporarily) as a secret key. A few bits of the secret key could be used as a pointer (*e.g.,* URL) to a place to post the opening of the commitment.

4.2 Implementation and use with Scantegrity

An interesting application of carbon dating is in end-to-end verifiable (E2E) elections. Scantegrity is an election system where the correctness of the tally can proven unconditionally [9], however this soundness relies, in part, on commitments made prior to the election. If a corrupt election authority changed the pre-election commitments after the election without being noticed, an incorrect tally could be made to verify. It is natural to assume that many people may only become interested in verifying an election after it is complete. Since the pivot (election day) is known, the commitments can be made well in advance, reducing the uncertainty of the carbon dating protocol. Moreover, owing to the design of Scantegrity, invalid commitments will only validate negligibly, ruling out precommitting to many possible values as an attack. Scantegrity was used in the 2011 municipal election in Takoma Park, MD (for a second time [8]) and CommitCoin was used to provide carbon dating of the pre-election commitments.[12]

References

1. T. Aura, P. Nikander, and J. Leiwo. DoS-resistant authentication with client puzzles. In *Security Protocols*, 2000.
2. A. Back. Hashcash: a denial of service counter-measure, 2002.
3. D. Bayer, S. A. Haber, and W. S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences*, 1991.
4. J. Benaloh and M. de Mare. Efficient broadcast time-stamping. Technical Report TR-MCS-91-1, Clarkson University, 1991.
5. J. Benaloh and M. de Mare. One-way accumulators: a decentralized alternative to digital signatures. In *EUROCRYPT*, 1993.
6. D. Boneh and M. Naor. Timed commitments. In *CRYPTO*, 2000.
7. A. Buldas, P. Laud, H. Lipmaa, and J. Villemson. Time-stamping with binary linking schemes. In *CRYPTO*, 1998.
8. R. T. Carback, D. Chaum, J. Clark, J. Conway, A. Essex, P. S. Hernson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, A. T. Sherman, and P. L. Vora. Scantegrity II municipal election at Takoma Park: the first E2E binding governmental election with ballot privacy. In *USENIX Security Symposium*, 2010.
9. D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman. Scantegrity II: end-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *EVT*, 2008.

[12] For full details, see `http://eprint.iacr.org/2011/677`

10. L. Chen, P. Morrissey, N. P. Smart, and B. Warinschi. Security notions and generic constructions for client puzzles. In *ASIACRYPT*, 2009.
11. J. Clark and U. Hengartner. On the use of financial data as a random beacon. In *EVT/WOTE*, 2010.
12. D. Dean and A. Subblefield. Using client puzzles to protect TLS. In *USENIX Security*, 2001.
13. S. Doshi, F. Monrose, and A. D. Rubin. Efficient memory bound puzzles using pattern databases. In *ACNS*, 2006.
14. C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, 1992.
15. M. K. Franklin and D. Malkhi. Auditable metering with lightweight security. In *Financial Cryptography*, 1997.
16. E. Gabber, M. Jakobsson, Y. Matias, and A. Mayer. Curbing junk e-mail via secure classification. In *Financial Cryptography*, 1998.
17. D. M. Goldschlag and S. G. Stubblebine. Publicly verifiable lotteries: Applications of delaying functions. In *Financial Cryptography*, 1998.
18. S. Haber and W. S. Stornetta. How to time-stamp a digital document. In *CRYPTO*, 1990.
19. M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In *Communications and Multimedia Security*, 1999.
20. A. Juels and J. Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *NDSS*, 1999.
21. G. O. Karame and S. Capkun. Low-cost client puzzles based on modular exponentiation. In *ESORICS*, 2010.
22. M. Mahmoody, T. Moran, and S. Vadhan. Time-lock puzzles in the random oracle model. In *CRYPTO*, 2011.
23. M. Mahmoody, S. P. Vadhan, and T. Moran. Non-interactive time-stamping and proofs of work in the random oracle model. IACR ePrint 553, 2011.
24. P. Maniatis and M. Baker. Enabling the long-term archival of signed documents through time stamping. In *FAST*, 2002.
25. T. Moran, R. Shaltiel, and A. Ta-Shma. Non-interactive timestamping in the bounded storage model. In *CRYPTO*, 2004.
26. S. Nakamoto. Bitcoin: A peer-to-peer electionic cash system. Unpublished, 2008.
27. B. Preneel, B. V. Rompay, J. J. Quisquater, H. Massias, and J. S. Avila. Design of a timestamping system. Technical Report WP3, TIMESEC Project, 1998.
28. R. L. Rivest and A. Shamir. PayWord and MicroMint: two simple micropayment schemes. In *Security Protocols*, 1996.
29. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report TR-684, MIT, 1996.
30. D. Stebila, L. Kuppusamy, J. Rangasamy, C. Boyd, and J. M. Gonzalez Nieto. Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols. In *CT-RSA*, 2011.
31. S. Tritilanunt, C. Boyd, E. Foo, and J. M. Gonzalez Nieto. Toward non-parallelizable client puzzles. In *CANS*, 2007.
32. X. Wang and M. K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *IEEE Symposium on Security and Privacy*, 2003.
33. B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for DoS resistance. In *CCS*, 2004.