Rational Secret Sharing with Side Information in Point-to-Point Networks via Time-Delayed Encryption

Anna Lysyanskaya
Brown University
anna@cs.brown.edu

Aaron Segal
Brown University
aaronak@cs.brown.edu

October 15, 2010

In this paper, we give the first construction of a rational secret sharing protocol that is strict Nash (or Nash with respect to trembles) in the computational sense, works in a standard point-to-point network with loose synchronization (i.e. does not rely on the availability of a broadcast channel), and can tolerate players with arbitrary side information about the secret. Since this has been proved impossible in the plain model, our protocol requires us to make assumptions about upper and lower bounds on the computational resources of the participants, and also to assume that they are out of sync with each other by at most a known quantity τ and that their network latency is at most some known quantity Δ. Specifically, we define and realize (in the random-oracle model, and assuming so-called standard architecture) time-delayed encryption, a primitive that allows a sender to create a ciphertext such that, for some parameter h, it will take $\Omega(2^h)$ time for the recipient to recover the plaintext, where, following the work on memory-bound functions, time is measured in memory accesses.

Rational parties will prefer to follow our protocol for reconstructing the secret even given a lot of side information about this secret. This was not true of any of the previously proposed strict Nash protocols for this task: In fact access to side information gave players in those protocols an incentive to deviate. As a result, for the first time, we have a rational secret reconstruction protocol that can be used in applications where the secret can be useful in the outside world (e.g. it can give players access to a valuable resource, or decrypt a file).

1 Introduction

In the last few years, the problem of rational secret sharing (RSS) and more broadly rational multi-party computation (RMPC) has become an active research topic [HT04, GK06, LT06, ADGH06, KN08a, KN08b, AL09, MNT09, FKN10]. Informally, the RSS problem is as follows: a set of rational (as opposed to honest or adversarial, as is the convention in cryptography) parties $P_1, \ldots, P_n$ each hold a share of a valuable secret s. Each of them is curious and wants to discover the secret s, and is also selfish and wants as few others as possible to be privy to s. A rational secret sharing protocol is one where $P_1, \ldots, P_n$ reconstruct s, such that none of them have an incentive to deviate.

To motivate the study of RSS, the following points are important: (1) it is a good idea to model protocol participants as rational rather than honest because this modeling is a better reflection of reality; (2) secret sharing is one of the most basic tasks for secure multi-party computation (MPC), and is in fact the building block in most general MPC protocols. Thus, before studying general rational MPC, it makes sense to understand rational secret sharing.

A formalization for what it means for participants to be motivated to follow a protocol, especially in the computational setting, has been the focus of much of recent work. The simplest and best understood solution concept is a *Nash equilibrium*. Informally, a protocol Π is a Nash equilibrium if P_i that believes that everyone else is following Π has nothing to gain by deviating from Π. Halpern and Teague [HT04] observed that it is not sufficient that a secret sharing protocol be a Nash equilibrium if we want a rational P_i to follow it. This is easy to see, using, for example, the following argument: Consider the case where $n-1$ shares are sufficient to compute s; further, consider the protocol Π in which each player P_i just broadcasts his share. If the other $n-1$ players follow the protocol, P_i has nothing to gain from deviating: everyone will see $n-1$ shares broadcast, and will be able to reconstruct the secret. But neither will P_i lose anything by deviating by staying silent: either way, he will receive the other players' shares. However, if another player P_j also stays quiet, P_i wins big by staying quiet: now only he and P_j are in a position to reconstruct s. If several other players stay quiet, then no one can reconstruct the secret, but even in that case P_i didn't lose anything by staying quiet: he would not have been able to reconstruct s even if he had broadcast his share. Therefore, even though Π is a Nash equilibrium, a rational player does no worse by deviating from it, and may in fact do better if not all other players follow the protocol. Hence it is in the player's best interest to deviate.

If a Nash equilibrium is not a sufficient condition to motivate rational players to follow a protocol, then what game-theoretic solution concept can give such motivation? And can a protocol satisfying a good enough solution concept be given in a realistic communication setting?

Starting with the work of Halpern and Teague [HT04], several papers suggested various protocols that survive iterated deletions of weakly dominated strategies [GK06, LT06, ADGH06]. Roughly, strategy σ'_i (such as the one where P_i stays silent in the example above) weakly dominates another strategy σ_i (such as the one where P_i broadcasts his share) if, depending on what other players do, P_i is never worse off, and sometimes better off, if he sticks to σ'_i. A strategy survives iterated deletion if it is not weakly dominated by anything, even after other weakly dominated strategies have been deleted (and there are an infinite number of iterations of these deletions).

To design a protocol that survives iterated deletion, the key idea, due to Halpern and Teague, is that the players don't know until after they've broadcasted their messages for a given round, whether or not this was the last round. As a result, they can be motivated not to deviate: deviation from the protocol can be detected and punished by the other players. Although working in slightly different models, all of the papers that focused on surviving iterated deletion required simultaneous broadcast, i.e. broadcast where Alice must send out her message for a given round before she receives Bob's messages for this round.

Although uncertainty about which round is last indeed fixed the problem that the $(n-1)$-out-of-n secret sharing example above suffers from, Kol and Naor [KN08b] demonstrated that just because a strategy survives iterated deletion does not mean that it is intuitively desirable; instead, they proposed the notion of a strict Nash equilibrium in which P_i has something to lose from deviating in the case when the others are following Π. The way to realize this concept was to ensure that in any given round, there is a *unique* message that P_i must send if he chooses to follow the protocol, and if he does not send it, then he visibly deviates and the other players can punish him. Moreover, their protocol did not rely on any cryptographic assumptions, and so for example was not susceptible to backwards induction [KN08a]. They also gave a version of their protocol that was strict ϵ-Nash without simultaneous broadcast; however, their protocols still needed a broadcast channel (but it didn't need to be simultaneous, in fact in their construction only one designated player would broadcast during any given round). Additionally, the way Kol and Naor treat side information is not suitable for many applications: as part of her share of the secret s, Alice received a vector of elements that was likely to contain s in the clear. In the absence of side information, Alice does not know for sure which element in her vector can be s (if any) and so is motivated to participate in the protocol. But if she has any side information about s, then she may be able to identify it among the vector of values she received and then she is best off not participating in the protocol at all.

The fact that a protocol does not allow the participants to have any side information about the secret s is inconvenient for applications. In particular, if s is the decryption key for a valuable encrypted file, or a password that gives you access to a valuable resource, then such a protocol cannot be used to have rational parties reconstruct s. But if s is just a random secret that has no value in the real world, then why would anyone be curious to learn it?

Among other contributions, Asharov and Lindell [AL09] showed that this limitation — no side information — is inherent whenever simultaneous broadcast cannot be assumed. Intuitively, this can be explained as follows: if some P_i can delay his round r messages until after he has received the round r messages from everyone else, then he can test if r is the last round by seeing what his output would be, if any, if it were the last round, and then checking that against his side information.

As for broadcast, Fuchsbauer, Katz and Naccache [FKN10] showed that this assumption could be dispensed with (but only did so in the computational setting, and achieving the computational versions of the appropriate equilibrium notions, also defined by Fuchsbauer, Katz and Naccache) and gave the first rational secret sharing protocols that did not require broadcast. Their protocols assumed that the players do not have access to any side information about the secret, however, and that all they know *a priori* is that s is a uniformly chosen binary string of a given length (known to all players).

In this paper, we restrict the model in a way that allows us to overcome the impossibility result of Asharov and Lindell: we give a rational secret sharing protocol that is a computational strict Nash equilibrium even in the presence of side information, and runs over a standard point-to-point network without requiring any form of broadcast. Instead of considering the standard model in which every player is an interactive probabilistic poly-time Turing machine (this is the model for which the impossibility result of Asharov and Lindell holds), we consider what happens when P_i is modeled in such a way that he cannot test whether round r is the last round fast enough to make the decision whether to send out his round r message or not. This is because in the last round all the players will output a *time-delayed encryption* of the secret s (introduced below), rather than s itself, and it will take each of them longer than the amount of time a round takes to decrypt it or in fact tell it apart from a (pseudo)random string — our protocol will essentially have the players output a pseudorandom string in every round other than the last one.

Time-delayed encryption. Motivated by this application to rational secret sharing, we define the new cryptographic primitive of a time-delayed encryption (TDE) scheme. The idea of TDE is to construct an encrypted message which can only be decrypted by its intended recipient, and even then cannot be decrypted until a moderate amount of time has elapsed. Although our primitive is a new one, we are not the first to consider such a problem; substantial work has been done in the field of timed-release cryptography (TRC), the goal of which is to encrypt a message so that it cannot be decrypted before a specific point in time [BCHVS08].

Past work in TRC is geared towards applications such as key escrow or protecting the secrecy of bids in a sealed auction, and is constructed with decryption delays of potentially days or years in mind. As such, existing TRC schemes often assume the presence of trusted authorities such as time servers [CHKO06], or identity-based encryption infrastructures [BF01]. By comparison, we need to ensure that the decryption of a message takes longer, but not a significant time longer, than a single round of a cryptographic protocol, and we cannot afford to assume the existence of trusted authorities.

The other major approach towards TRC, which does not rely on third parties, is the idea of the time-lock puzzle (TLP), first introduced by Rivest, Shamir, and Wagner [RSW96]. A TLP requires the decrypting party to perform a large computation in order to recover a decryption key; in most of the existing work on TRC, this computation is based on the non-parallelizable problem of repeated squaring modulo a product of two large primes. Although the notion of using a TLP to instantiate time-delayed encryption is attractive, using the repeated squaring problem is unrealistic for our application. Specifically, in order for a rational player to want to participate in our protocol, that player must be convinced that none of the others has a significant computational advantage in computing the TLP. If we design the TLP such that decrypting a message requires a large computational overhead, a rational player might not be willing to spend the resources to decrypt the message at all, preferring not to participate in the protocol. However, if the TLP takes only a moderate amount of time to solve, then a player running a computer with a very fast CPU might be able to decrypt the message well before a player with a slower computer is able to do so. In modern computers, the disparity between "fast" and "slow" CPUs is wide enough that a TLP relying on a CPU-bound algorithm like repeated squaring could not be fair for players with a range of technology available.

Our construction of the time-delayed encryption, both in the definition and in the construction, builds upon an alternative form of time-lock puzzle — a memory-bound function (MBF). Introduced by Dwork, Goldberg and Naor [DGN03] and studied further by Abadi et al. [ABMW05], MBF was motivated by proof-of-work applications to deter e-mail spammers. The justification for counting memory accesses rather than computational steps is that the speed of the former is more uniform throughout different architectures (see [DGN03] for a discussion on this).

How to construct a TDE from an MBF as originally defined by Dwork, Goldberg and Naor was not immediate, because their definition gave a lower bound on the expected number of memory accesses of an adversary/spammer, while we want to give a lower bound that will hold for every adversary with all but negligible probability. Thus, in adapting their work to our application, we needed to give a revised definition of MBF (called cryptographic MBF, or CMBF). We showed that the Dwork et al. construction of MBF, with very minor modifications, also satisfies our CMBF definition. Finally, we constructed a time-delayed encryption scheme from a CMBF. The lower bound on the number of memory accesses holds under the same assumptions on standard computer architecture as those made by Dwork et al.

Overview of the rest of this paper. We give a formal definitions of two (related) cryptographic primitives, a *time-delayed encryption scheme*, and a *time-delayed encryption scheme indistinguishable from random.* We then discuss and formalize two notions of a memory-bound, time-lock puzzle: the *memory-bound function* and the *cryptographic memory-bound function.* We show how to build a time-delayed encryption scheme indistinguishable from random by using a cryptographic memory-bound function treated as a black box, and prove that this construction satisfies our definition. We then show how to adapt an existing memory-bound algorithm to instantiate a cryptographic memory-bound function.

We then turn our focus to the problem of rational secret sharing in the presence of side information. We model side information as both a polynomially long string of information about the secret, and an oracle whose behavior may be dependent upon the secret (for example, it may output 1 only when given the correct secret as input), and discuss the game theoretic equilibria which motivate the construction of our protocol, specifically the notions of computational strict Nash equilibrium, and computational Nash stable with respect to trembles. We then give a full specification of our protocol, and analyze its performance with respect to these equilibria.

2 Time-Delayed Encryption

Consider the following problem: Alice has a message, m, which she wants to send to Bob. She is not concerned about security or demonstrating authenticity. Rather, she wishes to ensure that Bob must wait a moderate amount of time (say, a few seconds to a mintue) before he can read the message.

This problem motivates a cryptographic primitive which we call a *time-delayed encryption scheme.* We define this primitive in terms of a security parameter 1^k, and a hardness parameter h such that 2^h is polynomial (potentially quite large) in k, such that Alice can be sure that Bob will not recover m in less than $\Omega(2^h)$ steps. By a "step" here we mean a memory access on a "standard architecture" as defined by Dwork, Goldberg and Naor [DGN03]. We review their definition of a standard architecture in Appendix A.

Let Gen be an algorithm that, on input the security parameter 1^k and the hardness parameter h, outputs a key K, a sealed key $\tilde{K}$, and some additional information F that will be used in unsealing the key. Let Enc and Dec be encryption and decryption algorithms respectively, that are indexed by a key K output by Gen, such that $(\mathrm{Gen}, \mathrm{Enc}_K, \mathrm{Dec}_K)$ constitute an adaptively secure (under CPA and CCA2) symmetric cryptosystem. Let Unseal_F be an algorithm such that, if $(K, \tilde{K}, F)$ are the output of Gen, then $\mathrm{Unseal}_F(\tilde{K}) = K$. The time-delayed nature of the cryptosystem dictates that there is a lower bound of $\Omega(2^h)$ on the running time of Unseal_F, and that in fact this lower bound applies to any algorithm that computes the unsealed key K from $\tilde{K}$. More formally:

Definition 1 (Time-delayed encryption (TDE))**.** $(\mathrm{Gen}, \mathrm{Enc}_K, \mathrm{Dec}_K, \mathrm{Unseal}_F)$ with the properties described above is a *time-delayed encryption scheme* for message space M for security parameter k

and hardness parameter h if for any adversary $\mathcal{A}$ running on a standard architecture,[1] which runs in time polynomial in k and, in Step 4 of the following game, makes $o(2^h)$ memory accesses, the following experiment returns 1 with probability equal to $1/2 + \nu(k)$ for some negligible ν:

1. $(K, \tilde{K}, F) \leftarrow \mathrm{Gen}(1^k, h)$.
2. On input F, $\mathcal{A}$ runs for any polynomial in k number of steps, adaptively sending a set of queries $\{m_i, c_i\}$ and getting back $\{c_i' \leftarrow \mathrm{Enc}_K(m_i), m_i' \leftarrow \mathrm{Dec}_K(c_i)\}$ and then outputs two messages, m_0 and m_1, both in the message space M. The contents of $\mathcal{A}$'s cache and memory is saved.
3. A challenge ciphertext is generated: $c \leftarrow \mathrm{Enc}_K(m_b)$ for a random bit b.
4. $\mathcal{A}(c, \tilde{K})$ is restarted using the same cache and memory state where it stopped. It is allowed to make at most $o(2^h)$ memory accesses. $\mathcal{A}$ makes more queries as in Step 2. If $\mathcal{A}$ asks for the decryption of c, return 0. If $\mathcal{A}$ outputs b, return 1. Else, return 0.

For our application, we need a stronger notion: we need the challenge ciphertext c and the sealed key $\tilde{K}$ to be indistinguishable from random strings of the same lengths, even when the distinguisher (whose number of memory accesses is $o(2^h)$) is given the sealed key $\tilde{K}$. This is defined below. Note that Definition 2 implies Definition 1, while the converse is not true.

Definition 2 (TDE indistinguishable from random)**.** $(\mathrm{Gen}, \mathrm{Enc}_K, \mathrm{Dec}_K, \mathrm{Unseal}_F)$ with the properties described above is a *time-delayed encryption scheme indistinguishable from random* for message space M for security parameter k and hardness parameter h if for any adversary $\mathcal{A}$ running on a standard architecture, which runs in time polynomial in k and, in Step 4 of the following game, makes $o(2^h)$ memory accesses, the following experiment returns 1 with probability equal to $1/2 + \nu(k)$ for some negligible ν:

1. $(K, \tilde{K}, F) \leftarrow \mathrm{Gen}(1^k, h)$.
2. On input F, $\mathcal{A}$ runs for any polynomial in k number of steps, adaptively sending a set of queries $\{m_i, c_i\}$ and getting back $\{c_i' \leftarrow \mathrm{Enc}_K(m_i), m_i' \leftarrow \mathrm{Dec}_K(c_i)\}$ and then outputs a message $m \in M$. The contents of $\mathcal{A}$'s cache and memory is saved.
3. Two ciphertexts are generated: $c_0 \leftarrow \mathrm{Enc}_K(m)$, $c_1 \leftarrow \{0,1\}^{|c_0|}$. Two candidate sealed keys are set as follows: $\tilde{K}_0 = \tilde{K}$, $\tilde{K}_1 \leftarrow \{0,1\}^{|\tilde{K}|}$, and a random bit b is chosen.
4. $\mathcal{A}(c_b, \tilde{K}_b)$ is restarted using the same cache and memory state where it stopped. It is allowed to make at most $o(2^h)$ memory accesses. $\mathcal{A}$ makes more queries, as in Step 2. If $\mathcal{A}$ asks for the decryption of c, return 0. If $\mathcal{A}$ outputs b, return 1. Else, return 0.

2.1 Construction

In order to introduce a time delay, we use memory-bound functions [DGN03], which require a large number of memory accesses to compute. This is preferable to CPU-bound functions (which require a large number of operations) only in that memory access speed varies relatively little

[1]See Appendix A.

across architectures compared with CPU speed. (We refer the reader to Dwork et al. [DGN03] for a discussion on this issue.)

We give two definitions of memory-bound functions: the original definition [DGN03], and another definition that we need as a building block for our construction of a time-delayed encryption scheme indistinguishable from random.

Definition 3 ([DGN03])**.** A *memory-bound function* (MBF) family is a family $\mathcal{C}$ of deterministic functions such that an efficient (polynomial in security parameter k and with the number of memory accesses polynomial in the hardness parameter h) key generation algorithm outputs a key F for the function such that the evaluation of $\mathcal{C}_F$, when executed on a standard architecture for an arbitrarily long but finite number of times on random inputs, has an amortized complexity of $\Omega(2^h)$. More precisely: suppose a key F is chosen by the key generation algorithm with parameters k and h; suppose that, on input F, $\mathcal{A}$ outputs a set of tuples $\{(x_i, y_i)\}$ such that $x_i \neq x_j$ for all $i \neq j$ and for some polynomial p, $p(k)$ of these tuples are such that $y_i = \mathcal{C}_F(x_i)$. Then $\mathcal{A}$'s expected number of memory accesses, where the expectation is over the choice of F and the random choices made by $\mathcal{A}$, is $p(k)\Omega(2^h)$.

The construction of Dwork, Goldberg and Naor (DGN) [DGN03] satisfies this definition in the random-oracle model. Note, however, that this gives us only an amortized lower bound in expectation[2]. This definition works very well for the DGN application of fighting spam: to send a message x_i the adversary needs to invest some effort into computing $y_i = \mathcal{C}_F(x_i)$, and the total amount of effort has to be proportional to the number of messages sent out, so that the effort that went into computing y_i does not help with y_j. However, here the choice of the messages themselves is entirely up to the adversary.

This definition does not quite work for our purposes: instead of a lower bound on the adversary's memory accesses that holds in expectation, we need a lower bound that holds with overwhelming probability. Additionally, in our application, there will be a specific input x, chosen by the challenger, and we want an adversary making $o(2^h)$ memory accesses to be unable to compute $\mathcal{C}_F(x)$, even after investing a lot of time and memory accesses in the precomputation phase. Therefore, we give a revised definition.

Definition 4. A *cryptographic memory-bound function* (CMBF) family is family $\mathcal{C}$ of deterministic functions such that an efficient (polynomial in security parameter k and with the number of memory accesses polynomial in the hardness parameter h) key generation algorithm outputs a key F for the function such that the evaluation of $\mathcal{C}_F(x)$, when executed on a standard architecture, requires $\Omega(2^h)$ memory accesses whenever it computes $\mathcal{C}_F(x)$ correctly, even after a long pre-computation, except with probability negligible in k, where k is a security parameter, and the probability is taken over the choice of the key F and the choice of the input x.

More precisely: Suppose a key F is chosen by the key generation algorithm with parameters k and h; $\mathcal{A}$ is given F and is allowed to run for some polynomial in k number of steps, and can make any polynomial in k (so, potentially many more than 2^h which is only polynomial in k) number of memory operations. Finally, $\mathcal{A}$ signals that it is ready to be challenged, and receives a random challenge x, takes an additional poly number of steps during which it makes only $o(2^h)$ memory accesses, and outputs y. Then the probability that $y = \mathcal{C}_F(x)$ is upper-bounded by some negligible $\nu(k)$.

[2]The DGN paper does not actually give a formal definition of an MBF; our interpretation of their definition comes from their analysis of their construction.

Luckily, although the two definitions are incomparable[3], as we show in the proof of Theorem 2, in the random oracle model, the MBF of Dwork et al. [DGN03] satisfies this definition as well.

Now we give a construction, secure in the random-oracle model, of a time-delayed encryption indistinguishable from random using a CMBF family as a building block. For Enc and Dec, we can use any block cipher (strong PRP) with block and key length k, and just append a $k/2$-bit nonce to a $k/2$-bit message before encrypting (to give us adaptive CPA and CCA2 security). Note that the standard definition of a strong pseudorandom permutation implies that the resulting ciphertexts are indistinguishable from random strings, even in the presence of the Enc_K and Dec_K oracles, as long as the decryption oracle is not asked to decrypt a ciphertext that the adversary is trying to distinguish from random.

Let $\mathcal{C}$ be a CMBF that takes k-bit strings as input, and outputs strings of length k'. Let $H : \{0,1\}^{k'} \mapsto \{0,1\}^k$ be a function that will be modeled as a random oracle in our proof of security. We define Unseal_F as follows: $\mathrm{Unseal}_F(\tilde{K}) = H(\mathcal{C}_F(\tilde{K}))$. Finally, we define $\mathrm{Gen}(1^k, h)$ as follows: (1) generate $\tilde{K} \leftarrow \{0,1\}^k$; (2) generate F for the CMBF family $\mathcal{C}$; (3) compute $K \leftarrow \mathrm{Unseal}_F(\tilde{K})$; (4) output $(K, \tilde{K}, F)$.

Theorem 1. *The tuple* $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{Unseal})$ *as defined above is a time-delayed encryption indistinguishable from random for message space* $M = \{0,1\}^{k/2}$ *with security parameter* k *and hardness parameter* h.

Proof of Theorem 1. First, we note that the input-output behavior of the resulting algorithms is correct, and that $(\mathrm{Gen}, \mathrm{Enc}_K, \mathrm{Dec}_K)$ constitute a symmetric cryptosystem whose ciphertexts are indistinguishable from random strings, so just with oracle access to Enc_K and Dec_K, and without $\tilde{K}$, c is indistinguishable from random. $\tilde{K}$ was chosen at random, and so, by itself, is already indistinguishable from a random string. Our goal is to show that when viewed jointly by a memory-constrained adversary with access to Enc and Dec, c and $\tilde{K}$ are indistinguishable from random.

Next, we claim that, except with negligible probability, none of $\mathcal{A}$'s queries to its random oracle H will contain the value $\mathcal{C}_F(\tilde{K})$. To see that this claim is true, suppose the opposite. Then the following reduction $\mathcal{B}$ computes $\mathcal{C}_F(\tilde{x})$ with $o(2^h)$ memory accesses with black-box access to $\mathcal{A}$: on input F, generate a random key K. Start $\mathcal{A}$ on input F. Interact with $\mathcal{A}$ the way that the challenger for Definition 2 does: answer $\mathcal{A}$'s encryption and decryption queries using a random K as the key. Also answer $\mathcal{A}$'s queries to H by providing k-bit random strings as responses. Finally, $\mathcal{A}$ is ready to be challenged, and provides a message m. Compute $c \leftarrow \mathrm{Enc}_K(m)$, and signal to the CMBF challenger that $\mathcal{B}$ is ready to be challenged. Now $\mathcal{B}$ receives its challenge x and is not allowed to make more than $o(2^h)$ memory accesses (including those made by $\mathcal{A}$). It now sets $\tilde{K} = x$, and, implicitly (because it does not know the value $\mathcal{C}_F(x)$, in fact that's the value $\mathcal{B}$ wants to output), engineers the random oracle H so that $K = H(\mathcal{C}_F(x))$ (this is OK because K was chosen at random). Give $(c, \tilde{K})$ to $\mathcal{A}$, and from this point on, both $\mathcal{B}$ and $\mathcal{A}$ are allowed only $o(2^h)$ memory accesses. Continue answering $\mathcal{A}$'s encryption and decryption queries as directed by the challenger in Definition 2. Also answer $\mathcal{A}$'s queries to H, up until the i^{th} query for a randomly chosen i: when $\mathcal{A}$ makes its i^{th} query to H — let us say $\mathcal{A}$ asks for the value $H(y)$, — output this

[3] $\mathcal{C}_F$ can be a CMBF but not an MBF if it is easy to compute on the inputs x that start with many 0's: the CMBF challenger is unlikely to pick such an easy x as a challenge, but the MBF adversary will just output a list that includes all the easy x's. $\mathcal{C}_F$ can be an MBF but not a CMBF if there is a noticeable fraction of bad keys F. It is actually possible to construct a CMBF from an MBF by picking many keys $\mathbf{F} = F_1, \ldots, F_\ell$ and setting $\mathcal{C}_{\mathbf{F}}(x) = \{\mathcal{C}_{F_i}(x)\}$, but we took the (arguably) simpler and more efficient route of showing that the DGN MBF is also a CMBF.

y. If, with non-negligible probability $\epsilon(k)$, one of $\mathcal{A}$'s $p(k)$ (for some polynomial p) queries was for the value $\mathcal{C}_F(\tilde{K})$, then with probability $\epsilon(k)/p(k)$ (which is still non-negligible), $\mathcal{B}$ correctly guessed the index i of this query, and thus outputs $y = \mathcal{C}_F(\tilde{x})$, even though, like $\mathcal{A}$, it made only $o(2^h)$ memory accesses once its challenge was issued. This contradicts the definition of a CMBF.

Note that, if $\mathcal{A}$ never queries H for the value $\mathcal{C}_F(\tilde{K})$, then $K = H(\mathcal{C}_F(\tilde{K}))$ is in fact independent of $\tilde{K}$ given the view of $\mathcal{A}$, or, equivalently, $\tilde{K}$ is a random string that is independent of $\mathcal{A}$'s Enc_K and Dec_K oracles and challenge ciphertext c. Therefore, $(c, \tilde{K})$ are indistinguishable from random when viewed together and with access to Enc_K and Dec_K (other than for querying Dec_K on the challenge ciphertext c). $\square$

2.2 Adapting the DGN Construction

Here, we show that the original DGN construction of an MBF also satisfies our definition of a CMBF.

The DGN memory-bound algorithm was designed as a countermeasure against spammers. Keyed by F, it takes as input the e-mail message, date, sender, and recipient, and returns a valid (not necessarily *unique*) "proof of work" for these inputs, that is to say, proof that the sender has spent some time on computing his message, so that it may be worth the recipient's while to look at it. We modify this algorithm only slightly. Again, the algorithm is keyed by F.

We specify that all strings of length $|\tilde{K}|$ are valid inputs, and only accept one specific response as a valid output for each input. Making no other changes to the DGN algorithm apart from these, we prove below that our adapted function is a cryptographic memory-bound function. Our CMBF takes as input a string x of length $|\tilde{K}|$, and returns a trial number t and a string y. H_0, H_1, H_2, and H_3 are random oracles. A is a modifiable array of size $|A|w > b$ bits, where w is the number of bits in a word, and b is the number of bits in a cache line (that is, the number of bits that are read in one memory access).

Theorem 2. *Given the following conditions, Algorithm 1 is a cryptographic memory-bound function for security parameter k and hardness parameter h:* (1) $|F| \geq 2s$ *(where the adversary's cache contains s words of w bits each),* (2) $z = h + k$, (3) $|A|w > 68.25b$ *(where b is the size of a cache line, in bits), and* (4) $l > 40|A|$.

We note that the precise values in the third and fourth conditions above are due only to the constants we use in our proof; for different constants, these values may vary. Our proof of this theorem closely follows the DGN proof that their construction (this same construction) is an MBF.

Proof of Theorem 2. We start with a simple lemma about the number of calls to H_1 made by the adversary.

Lemma 1. *Except with probability negligible in k, the number of calls to H_1 necessary to determine the first successful path is $\Omega(2^h l)$.*

Proof of Lemma 1. Since H_1 and H_2 are random oracles, it takes ℓ calls to H_1 to determine whether a given path is successful. Since H_3 is a random oracle, the probability that any given path is successful is 2^{-z}. We want to use a Chernoff bound to give us the probability that, after looking at only 2^h paths, we will have at least 1 successful path. We get that this probability is at most $2^{h-z}e^{1-2^{h-z}}$. Since we took $z = h + k$, this gives us $2^{-k}e^{1-2^{-k}} \leq 2^{-k}e$. $\square$

Setup: Let F be a random function from the set $\{0, 1, \ldots, \#F\}$ to the set of w-bit integers. Once F is chosen, it is fixed for the entire execution of the memory-bound function, and may be reused for future executions of the same function. (F can also be thought of as a table containing $\#F$ words of size w bits — we use these two notions interchangeably.) Also choose ℓ and z, parameters which control the amount of effort required to calculate the function.
Algorithm: On input x, do the following: For $t = 1, 2, \ldots$:

```
• $A \leftarrow H_0(x, t)$
• Repeat $l$ times:
      $q \leftarrow H_1(A)$
      $A \leftarrow H_2(A, F(q))$
• $y \leftarrow H_3(A)$
• If the last $z$ bits of $y$ are all zero, return $t, y$ and halt. Else, continue with $t + 1$.
```

Each complete walk through the outer loop is called a *path*. We say that a path number t is *successful* if, at the end of the path, the last z bits of $H_3(A)$ are all zero. Note that there may be more than one successful path, but only the first successful path is the correct output from our function.
We remark that the paths are independent of each other (that is, the initialization of the path depends only on the function input x and the path number). This property allows successful paths to be verified in only $O(l)$ memory accesses, which is crucial for anti-spam applications [DGN03]. However, the independence of the paths allow for trivial parallelization of the algorithm, which could allow the function to be computed too quickly. The algorithm can be made non-parallelizable by substituting A for x in each call to H_0, except in the first path of the algorithm.

Figure 1: CMBF based on the DGN MBF

We next present a purely combinatorial lemma which we use in our proof.

Lemma 2. *Let the set S be constructed by picking n integers uniformly (with replacement) from the range $[1, m]$. Then the probability that S contains at most k distinct values is negligible in n, as long as $0 < k < n < m$ and $1.5nk/m \leq n - k$.*

Proof of Lemma 2. Consider the game (which we will call G_1) in which we draw the integers one at a time; after n draws, the game is lost if there are at most k distinct values among the integers drawn, and won otherwise. Equivalently, the game is lost if at least $n - k$ of the draws are repeats of previous draws.
Let an instance of G_1 be described as an n-tuple of integers in $[1, m]$; a run of G_1 is equivalent to uniformly choosing one of these m^n n-tuples. We give a permutation from instances of G_1 into instances of another game G_2, which is in some sense "harder" than G_1. We will then argue that the probability of losing G_1 is no more than the probability of losing G_2, and that the probability of losing G_2 is negligible. This will give us the desired result, that the chance of losing G_1 is negligible.
We first specify G_2. Let G_2 be the following game: Draw n integers uniformly one at a time (with replacement) from $1, \ldots, m$. If, during the game, at least $n - k$ draws are integers between 1 and k, inclusive, then the game is lost. Otherwise, the game is won. Since the n integers are chosen independently, each has a k/m probability of being in the range from 1 to k. In expectation, the number of integers in the range is nk/m, and using a Chernoff bound, we get that

the probability that at least $1.5nk/m$ integers fall in the range is $(8e/27)^{kn/2m}$. Therefore, as long as $1.5nk/m \leq n-k$, the probability of losing H_2 is negligible in n. Now, we have to construct the permutation from games of G_1 to games of G_2. We define an instance of G_2 analogously to instances of G_1; both types of instances are specified by n-tuples of integers from 1 to m. We also note that any run of either game has a m^-n chance of being each of these n-tuples. Therefore, if the number of n-tuples that are losing for G_1 as at most the number n-tuples that are losing for G_2, then the probability of losing a game of G_1 is at most the probability of losing a game of G_2, which we have shown is negligible.
In order to ensure that there are at least as many losing instances of G_2 as of G_1, our permutation will enforce that every losing instance of G_1 is mapped to a losing instance of G_2. It may map some winning instances of G_1 to losing instances of G_2 as well, but this is acceptable since all we need is an upper bound.
Our permutation will work as follows. To convert a run of G_1 into a run of G_2, we step one at a time through each integer drawn in G_1, and decide which integer was drawn in the corresponding instance of G_2. During this process, we will maintain a set S of unique integers drawn in G_1, and a modifiable array M of m integers in $[1, m]$, which will give us a mapping from numbers drawn in G_1 to integers drawn in G_2. Whenever an integer a is drawn in G_1, we will say that the integer $M[a]$ has been drawn in G_2. In between draws, we may modify M by swapping two entries in it.
Initialize $S = \emptyset$, and initialize M by setting $M[i] = i$ for each i from 1 to m. We then repeat the following process n times:

1. Let a be the next integer that was drawn in G_1.
2. $M[a]$ is the next integer that is drawn in the corresponding instance of G_2.
3. If $a \notin S$, add a to S, and then swap $M[|S|]$ and $M[a]$. (Note that $|S|$ starts at 0, and is incremented by 1 every time we draw a new unique element.)

To see why every losing game of G_1 maps to a losing game of G_2 by this process, remember that a losing instance of G_1 is one in which at least $n-k$ of the draws are repeats of previous numbers drawn, and a losing instance of G_2 is one in which at least $n-k$ of the draws are numbers between 1 and k. Our permutation ensures that, as long as there are fewer than k distinct elements drawn in G_1, every draw in G_1 which counts against our $n-k$ repeats is a draw in G_2 which counts against our $n-k$ numbers $\leq k$.
Finally, we need to show that our transformation of G_1 games into G_2 games is, in fact, a permutation. To do this, we exhibit the reverse transformation of G_2 games into G_1 games. This reverse transformation will be very similar to our forward one. It will go step by step as in the forward transformation, and will maintain a set S' and an array M' of m numbers from 1 to m.
Initialize $S' = \emptyset$, and initialize M' by setting $M'[i] = i$ for each i from 1 to m. Repeat the following process n times:

1. Let a be the next integer that was drawn in G_2.
2. $M'[a]$ is the next integer that is drawn in the corresponding instance of G_1.
3. If $M'[a] \notin S$, add $M'[a]$ to S', and then swap $M'[|S'|]$ and $M'[M'[a]]$.

Suppose we ran the forward transformation on a game of G_1, and at the same time, we ran the reverse transformation on the draws of G_2 recorded by the forward transformation. At the end of each step, $S = S'$, and $M'[M[i]] = i$ for all i from 1 to m. Therefore, our transformation is a permutation, as required. $\square$

We now want consider the adversary's program, broken up into intervals. Similarly to [DGN03], we will show that, during each of these intervals, the adversary must make a large number of memory reads. Since each A is the output of a random oracle, we want to think of it as incompressible. However, the adversary could just store the values (x, t) associated with the beginning of the path, and reconstruct A when needed. If the A in question came from the second half of a path, however, it would be less work for the adversary to finish the path than to restart the same path. We will therefore count only accesses to H_1 made in the second half of a path when determining the length of an interval.

Let a *mature progress call* be defined as a call to H_1 such that at least $l/2$ calls to H_1 have already been made on the same path. Define $n := s/|A|$; that is, n is the number of A's that can fit in the cache at once. Define an *interval* by picking an arbitrary point in the execution of the adversary's program, and continuing until $8n$ mature progress calls have been made.

Lemma 3. *The number of memory accesses made during any given interval is $\Omega(n)$ except with negligible probability.*

By Lemma 1, the number of intervals is $\Omega(2^h\ell/n)$. Multiplying this number of intervals by $\Omega(n)$ memory accesses per interval gives us $\Omega(2^h\ell)$ memory accesses in total, and we will have our theorem. The rest of our proof, therefore, is given over to proving Lemma 3.

Proof of Lemma 3. Since we require $|F| \geq 2s$ (where s is the number of words in the cache), intuitively, at least half of the bits in F are missing from the cache at any time.

Claim 1. *There exist $\gamma, \delta \geq 1/2$ and a set $F' \subseteq F$ such that $|F'| = \delta|F|$ and, given the cache contents at the beginning of the interval, for each entry i in F' there is a set S_i of $2^{\gamma w}$ possible values for $F[i]$, and all the S_i's are mutually consistent with the contents of the cache.*

Proof. Since F is randomly generated, it is incompressible; that is, it takes $|F|w$ bits of information to specify the entire table. So, if the cache contains s words of size w, at most sw bits of information about F are known, leaving at least $(|F|-s)w \geq (|F|/2)w$ bits of information about F still unknown.

Say the adversary wanted to specify as many entries in F using only sw bits. Assuming F is incompressible, at most $s \leq |F|/2$ entries could be completely specified in this way, leaving at least $|F|/2$ entries of F completely unknown at the start of the interval. Alternatively, the sw bits could be used to give $w/2$ bits of information about at most $2s \leq |F|$ entries in F, leaving at least $w/2$ bits of information unknown (and thus at least $2^{w/2}$ valid possibilities) for every entry in F. We can therefore say that there are $2^{\gamma w} \geq 2^{w/2}$ valid possibilities for at least $\delta|F| > |F|/2$ of the entries in F that are consistent with the information in the cache. This is a generous lower bound, but it captures all possibilities in which the adversary, at the start of the interval, knows all of some entries of F and some bits of others. $\square$

Claim 2. *If the number of memory accesses during an interval is $o(n)$, then the number of different paths on which a mature progress call can be made during an interval is at most $3n$ (except with probability negligible in n).*

Proof of Theorem 2. In order to make a mature progress call, the adversary must know the value A. This A could have been known at the start of the interval, or a previous A from earlier in the same path was known at the start of the interval, or the entire first half of the path was developed during this interval. Consider all the paths for which a previous A was known at the start of the interval, and suppose for the sake of contradiction that there are at least $2n$ such paths. In order to make at least one call along each path, the adversary requires $2n|A|w$ bits of information. Since the cache only holds $s = n|A|$ words, only $n|A|w$ bits of this information can possibly be in the cache at the start of the interval, so the adversary needs to bring the other $n|A|w$ bits into the cache from main memory. Since we requried that $|A|w > b$, where b is the size of a single block, this requires $\Omega(n)$ memory reads, contradicting our assumption that the number of memory reads is $o(n)$.

Now, consider all the paths for which no previous A was in the cache at the start of the interval. Suppose the adversary can make at least one mature progress call on each of n such paths. This requires that all n paths first be brought to maturity, which requires $nl/2$ accesses to F. We argued above that there exists a subset of the entries of F, called F', which is missing almost fully from the cache at the beginning of the interval, and that $|F'| \geq 1/2|F|$. Since H_1 is a random oracle, we say that each of the $nl/2$ (possibly repeating) elements we need is chosen uniformly (with possible repetitions) from the entries in F. Consider the event that all of these elements fall into a set of size at most $5/8|F|$. By Lemma 2, the probability of this event is negligible in n as long as $(3/2)(nl/2)(5/8|F|)/(|F|) < 3/8|F|$, or $20|F| < nl$. Since $|F| \geq 2s$ and $n = s/|A|$, this is guaranteed if $l > 40|A|$, which we assume in our statement of the theorem.

Except with negligible probability, $5/8$ of the elements of F will be required to start n new paths. We argued above that at least half the elements of F are missing almost fully from the cache at the start of the interval, leaving at least $1/8|F|$ elements of $|F'|$ to be learned during the interval. The adversary therefore needs to bring at least $|F|w/8 = n|A|w/4$ bits into the cache from main memory, requiring $\Omega(n)$ memory reads and contradicting our assumption. □

Claim 3. *Except with probability negligible in n, the adversary must use an entry from F' to make a call to H_2 $\Omega(n)$ times during an interval.*

Proof. During the interval, the adversary needs to make $8n$ calls to H_1. In order to keep making progress on a path, the adversary must follow each call to H_1 by a call to H_2, which requires accessing an element of the table F. Since the elements of F' are missing from the cache, and $|F'| \geq 1/2|F|$, each call to H_2 has probability at least $1/2$ of requiring an element of F'. For each element, the adversary has the choice whether or not to read it from main memory during this interval, or not. If not, that call to H_2 is not made, and no further progress along that particular path can be made this interval. However, since we argued that the adversary can explore no more than $3n$ paths during the interval, the adversary can refuse to learn an element of F' (which would halt progress on a path) no more than $3n$ times. In expectation, the adversary will require at least $4n - 3n = n$ words from F' during any given interval. By applying the Chernoff bound for symmetric random variables, we get that the probability that at most $3.5n$ repsonses from H_1 will require elements of F' is $e^{-n/16}$, which is negligible in n and would still require $n/2 = \Omega(n)$ elements of F'. □

The adversary must, except with negligible in n probability, learn at least $n/2$ elements of F' in order to complete the interval. However, it is possible that there are $n/2$ elements of F' are stored close together in main memory, so that the adversary only needs to do $o(n)$ memory reads in order to learn all $n/2$ words. This would require that the $3.5n$ elements of F', selected randomly by H_1 during the interval, fall into a relatively small number of blocks, say at most $3.25n$ blocks. The adversary could ignore no more than $3n$ of these blocks in order to continue making progress on at least one path, leaving $n/4 = \Omega(n)$ blocks that need to be read.

Define $m := |F'|w/b$; since b is the number of bits in a block of memory, this is the minimum number of blocks into which the words in F' can be stored. Suppose that the words in F' are indeed stored in exactly m blocks; number these 1 through m. Since H_1 is a random oracle, we can say that each of the $3.5n$ elements of F' returned by H_1 is stored in block i with probability $1/m$, for $1 \le i \le m$. By Lemma 2, the probability that the $3.5n$ elements of F' fall into at most $3.25n$ blocks is negligible in n, as long as $(1.5)(3.5n)(3.25n)/m \le n/4$, which works out to $68.25n < m$. We have $m = |F'|w/b \ge sw/b$, and $n = s/|A|$, so our requirement is satisfied as long as $|A|w > 68.25b$, which is assumed in the statement of our theorem.

Therefore, the probability that the adversary can complete the interval in $o(n)$ memory reads is negligible in n, completing our proof of Lemma 3 (and Theorem 2). □

□

3 Rational Secret Sharing

A t-out-of-n secret sharing protocol consists of a secret sharing phase, and a reconstruction phase. In the secret sharing phase, a dealer D takes as input some value s from some well-defined domain, a security parameter 1^k, the number of parties n and parameter t and outputs values $s_1, \ldots, s_n$, such that no subset of fewer than t of these values reveals any information about the secret s. In the reconstruction phase, each party P_i is given s_i and the security parameter and they run (potentially in the presence of an adversary that can control up to $t-1$ of them and/or that prevents participation of some additional honest parties) some secure multi-party protocol as a result of which they reconstruct s whenever t of them follow the protocol (and the rest may possibly deviate). (We omit the formal definition here.)

In the setting, introduced by Halpern and Teague [HT04], where there is no adversary, but each P_i is rational rather than honest, we need to design protocols where no participant will have an incentive to deviate. To that end, we first need to model the payoffs the participants receive depending on whether or not they learn the secret s, or in fact any partial information about this secret. Following Halpern and Teague [HT04], let us assume that the following holds about the utility function μ_i of participants P_i: we assume that P_i strictly prefers any outcome in which he outputs s to any outcome where he does not, and, additionally, whether he output s or not, he strictly prefers the outcome where k others have output s to the one where $k+1$ have, for $0 \le k < n$.

We use standard [HT04] notation for talking about strategies and their expected utilities. Let $\sigma = \{\sigma_i : 1 \le i \le n\}$ be a vector of strategies for the players $\mathbf{P} = \{P_i : 1 \le i \le n\}$. Then $u_i(\sigma)$ denotes the expected utility P_i receives when each P_j runs σ_j. This expectation is taken over the choice of the secret s itself as well as the random choices of the dealer and each strategy. Further, (σ'_i, σ_{-i}) denotes the vector of strategies obtained from σ by replacing σ_i with σ'_i. By $u_i(\sigma)$, we denote the utility derived by P_i when everyone follows the set of strategies σ.

Our goal is to design a protocol Π that a rational participant will prefer to follow, rather than to deviate from. To that end, we will design a protocol Π that induces a computational strict Nash equilibrium that is stable with respect to trembles, essentially as defined by Fuchsbauer, Katz and Naccache [FKN10], but with some modifications (discussed below). We refer the reader to their work for a definitional discussion, and below we simply reproduce their definition and explain why our modifications were required. We note that, although our protocols work in the timing model, they can still be analyzed with respect to these definitions; i.e. the game-theoretic notions need not be revised due to the change of the model.

However, the timing model does allow us to achieve these forms of equilibrium even when some players have access to arbitrary side information about the secret, something that cannot be achieved in the standard model [AL09].

Corresponding to each player P_i, let aux_i be any potentially randomized function whose output length is polynomial in the length of its input. Further, let $\mathcal{O}_s^i$ be an oracle whose behavior may be dependent on s (for example, it may output 1 on input s, and 0 on all other inputs; or it may output 1 on input s' if $f(s) = f(s')$ for some f, etc). We say that a player P_i is *interested is the publication of* s in the presence of $aux_1(s), \ldots, aux_n(s)$ and $\mathcal{O}_s^1, \ldots, \mathcal{O}_s^n$ if, when each P_j is given $aux_j(s)$ and oracle $\mathcal{O}_s^j$, P_i derives more utility from the scenario where s is published than from the one where nothing further about s is communicated to any participant.

We must restrict our attention to the situation where each participant is interested in the publication of s, because a rational P who is not interested in the publication of s is not going to participate in a secret reconstruction protocol: he would rather no reconstruction took place. An example of when P_i is not interested would be if his auxiliary information $aux_i(s)$ narrows down the choices of s to a small number of candidates, and his oracle $\mathcal{O}_s^i$ outputs 1 on input s and 0 on all other inputs. In this case, P_i can learn s on his own and gain more utility, because he prefers that as few as possible players learn s. Following Fuchsbauer et al., we focus on a somewhat simplified setting where, in the presence of the auxiliary information and oracles for each player, if P_i outputs the secret and no other player does, then he receives utility U^+, if he outputs the secret and so does some other player, then his utility is U, while if he does not output it then it is U^-, and $U^+ > U > U^-$.

Since we allow players to have side information about s, a player may, with some probability, correctly guess s based only on his side information (that is, without running the protocol). Let $U_i^{guess} = U^+ \Pr[P_i \text{ correctly guesses } s] + U^-(1 - \Pr[P_i \text{ correctly guesses } s])$. Assuming P_i is interested in the publication of s, then $U_i^{guess} < U$.

An additional restriction on the oracles $\mathcal{O}_s^i$ is that their responses to every query must be poly-time computable from s. This is because otherwise they can break security of our cryptographic primitives.

By $u_i((\sigma_1, \ldots, \sigma_n), (aux_1, \ldots, aux_n), (\mathcal{O}_1, \ldots, \mathcal{O}_n))$, we denote the utility derived by P_i when every player P_j, $1 \le j \le n$, follows strategy σ_j with side information $(aux_j, \mathcal{O}_j)$.

3.1 Game Theoretic Equilibria

Our goal is to provide a protocol (also frequently referred to as a strategy) such that no rational party has an incentive to deviate from it. A variety of notions proposed in the literature are a formalization of this idea, notably (1) that of a Nash equilibrium where, assuming everyone else is following a prescribed protocol, no player has anything to gain by deviating from it; (2) that of *computational* Nash equilibrium [DHR00] where, assuming everyone else is following a prescribed protocol, a polynomial-time player can only gain at most a negligible amount by deviating from it; (3) that of *(computational) strict* Nash equilibrium [KN08a, FKN10] where, assuming everyone else is following the protocol, (with all but negligible probability a poly-time) player has a non-negligible amount to lose from deviating; (4) that of computational Nash equilibrium that is stable with respect to trembles [Kat08, FKN10], where assuming everyone else is following the protocol with high probability (and is deviating in an arbitrary way with some low, although noticeable, probability), no player has anything to gain by deviating; (5) that of a strategy that survives the process of iterated deletions of weakly dominated strategies, that is strategies for which another strategy is better in some circumstances and at least as good in others [HT04, KN08a].

Following Kol and Naor and Fuchsbauer et al., we focus on the notions of strict computational Nash and computational Nash stable with respect to trembles. As discussed in the introduction, a non-strict equilibrium may still provide an incentive to deviate if a player believes that other players may deviate. Further complicating the issue, in the computational setting each player may want to deviate anyway, for example so that he can compute more information by investing more time into the computation (see, for example, Lysyanskaya and Triandopoulos [LT06] for a discussion on this issue); Fuchsbauer et al. get around this by only comparing a given strategy to strategies that visibly deviate from it, so that essentially a strategy that carries out some additional computation on the side is considered equivalent to one that does not, more on this definition below. Surviving iterated deletion seemed like a good indication that a strategy does not suffer from such shortcomings; however, as shown by Kol and Naor, this notion is useless in a setting where no strategy weakly dominates another and so every strategy survives.

We now give a modified definitions of strict computational Nash equilibrium which takes into account that in our setting each player has access to auxiliary information and, additionally, a side information oracle. Here, let Π be a secret sharing scheme and σ_i be the prescribed strategy of P_i in the reconstruction phase.

Definition 5 (Computational Nash with side information)**.** Π induces a computational Nash equilibrium with side information $(AUX, \mathcal{O}) = \{aux_i, \mathcal{O}_i\}$ if for every P_i, any probabilistic polynomial-time strategy σ'_i, $u_i((\sigma'_j, \sigma_{-j}), AUX, \mathcal{O}) \leq u_i(\sigma, AUX, \mathcal{O}) + \nu(k)$ for some negligible ν.

Fuchsbauer et al. defined the notion of equivalent play: let Π be a protocol for $P_1, \ldots, P_n$, and let ρ_i be a strategy for P_i. Then ρ_i yields equivalent play with respect to Π (denoted $\rho_i \approx \Pi$) if given the views of all the players $P_j \neq P_i$, it is impossible to distinguish, in polynomial time, whether P_i is following Π or some prefix of ρ_i. (By "prefix" of ρ_i we mean that P_i was following ρ_i but at some point stopped sending any messages to other players.) We augment their definition to that of equivalent play with side information: Let $(AUX, \mathcal{O}) = \{aux_i, \mathcal{O}_i\}$ be the side information available to the players; then $\rho_i \approx_{(AUX,\mathcal{O})} \Pi$ if, given the views of all the players $P_j \neq P_i$, including their auxiliary information and all of their oracle queries, no polynomial-time algorithm can distinguish whether P_i is following Π or some prefix of ρ_i. (Note that we've omitted some of the details of this definition; we refer the reader to Fuchsbauer et al. for full details of their definition, and note that our modifications are relatively straightforward.)

Definition 6 (Computational strict Nash with side information)**.** Π induces a computational strict Nash equilibrium with side information $(AUX, \mathcal{O}) = \{aux_i, \mathcal{O}_i\}$ if it is a Nash equilibrium with side information and for every P_i, for any probabilistic polynomial-time strategy $\sigma'_i \not\approx_{(AUX,\mathcal{O})} \Pi$, $u_i((\sigma'_i, \sigma_{-i}), AUX, \mathcal{O}) < u_i(\sigma, AUX, \mathcal{O}) + \epsilon(k)$ for some non-negligible ϵ.

The idea of a protocol that is stable with respect to trembles is motivated by Katz [Kat08], and defined by Fuchsbauer et al. [FKN10]. It captures the idea that if P_i believes that the other players are most likely following Π, then P_i has nothing to gain from deviating – even if with some probability they are following some other strategy, in fact some strategy designed in such a way as to induce P_i to deviate. For $0 \leq \delta \leq 1$, $1 \leq i \leq n$, let $\mathbf{\Sigma}^{\delta}_{-i}$ be the set of strategies where with probability $1-\delta$ the players $P_j \neq P_i$ follow $\Pi = (\sigma_1, \dots, \sigma_n)$, and with probability δ they do something else.

Definition 7 (Computational Nash stable wrt trembles with side information)**.** Π induces a computational Nash equilibrium stable with respect to trembles with side information $(AUX, \mathcal{O}) = \{aux_i, \mathcal{O}_i\}$ if it is a Nash equilibrium with side information and for every P_i, for all non-negligible δ, for all $\sigma'_{-i} \in \mathbf{\Sigma}^{\delta}_{-i}$, for any $\sigma'_i \not\approx_{(AUX,\mathcal{O})} \Pi$, $u_i((\sigma'_i, \sigma'_{-i}), AUX, \mathcal{O}) \leq u_i((\sigma_i, \sigma'_{-i}), AUX, \mathcal{O}) + \nu(k)$ for some negligible ν.

3.2 The Protocol

Intuition. Our protocol builds on that of Fuchsbauer et al. [FKN10], but is a strict computational Nash with respect to trembles even if all or some of the participants P_i have access to a side information oracle $\mathcal{O}^i_s$ or know other side information about s captured by a string aux_i.

Fuchsbauer et al.'s protocol works as follows: the dealer picks a round r^* using the geometric distribution, such that in this round, the participants will reconstruct the secret s, and in the following round r^*+1, they will learn that the value they just reconstructed was indeed s. A participant ending the protocol or deviating from it after just learning the outcome s_r of a particular round r does not know whether $s_r = s$, and has no way of finding that out, so he prefers to continue participating in the protocol until he learns for sure that a particular value was correct. Of course, this analysis crucially relies on the player's inability to verify whether s_r is the correct secret.

Access to side information (especially side information that the dealer may not have had and had no way of accounting for, i.e. no way to create a believable-looking but still incorrect s_r) may dramatically alter a player's payoffs, and may make deviation upon learning s_r desirable.

For example, suppose that s is a decryption key for a publicly available ciphertext, or a signing key corresponding to a publicly available verification key. Then all players have sufficient side information to recognize s when they see it, so the Fuchsbauer et al. results don't apply. What is more, the impossibility result of Asharov and Lindell [AL09] showed that in the standard model, this limitation is inherent.

Our main modification of the Fuchsbauer et al. protocol is to have the player learn $s_{r^*} = \tilde{s}$, which is a time-delayed encryption of s, in round r^*, and a random string s_r in every round $r \neq r^*$. By introducing timed rounds, we force each participant to make the choice to either follow the protocol or deviate from it *before* any of them can determine the contents of $\tilde{s}$ and check if it's the secret s. Thus, deviation upon learning s_r is not desirable: no P_i has time to figure out if $s_r = \tilde{s}$ or is just a random string, and so needs to play along so as not to disrupt the protocol: a P_i that deviates from the protocol before obtaining $\tilde{s}$ will never learn s.

Building blocks. Our protocol requires verifiable random functions (VRFs), which were introduced by Micali, Rabin, and Vadhan [MVR99]. Here, we need VRFs with unique proofs; constructions of such VRFs in the standard model have been given by [Dod03] and [DY05], among others.

Recall that a VRF is a pseudo-random function, augmented with a public key and a secret key, such that the function cannot be computed without the secret key, but input-output pairs can be verified with the public key (and some proof information, generated with the secret key).

More formally a VRF is a tuple of probabilistic, polynomial-time algorithms (GenVRF, Eval, Prove, Vrfy), such that for any (PK, SK) output by $\text{GenVRF}(1^k)$, we have:

Pseudorandomness Eval_{SK} is a PRF

Correctness $\text{Vrfy}_{PK}(x, \text{Eval}_{SK}(x), \text{Prove}_{SK}(x)) = 1$

Verifiablilty There does not exist a tuple (x, y, y', π, π') with $y = y'$ and $\text{Vrfy}_{PK}(x, y, \pi) = 1 = \text{Vrfy}_{PK}(x, y', \pi')$

Unique proofs There does not exist a tuple (x, y, π, π') with $\pi = \pi'$ and $\text{Vrfy}_{PK}(x, y, \pi) = 1 = \text{Vrfy}_{PK}(x, y, \pi')$

Timing assumptions. First, let us assume that all participants have a realistic idea of the clock drift, network latencies, and each others' speeds. In other words, all participants have agreed on some values τ, Δ, speed_{max} that are upper bounds on the actual clock drift, network latency, and speed of each party, respectively. All participants also agree on some value speed_{min} which is a lower bound on their true speed of computation. The equilibrium properties of our protocol crucially rely on the assumption that all parties have access to these values.

Note that if at time t, P_j is supposed to send a message to P_i, then P_i knows that by the time his local clock shows $t + \tau + \Delta$, he should have heard from P_j if P_j is indeed following the protocol. Note also that, if a particular subroutine of the protocol requires k computation steps and is scheduled to start at time t, then at time $t + k/\mathsf{speed}_{min}$ everyone will have completed it.

The Timing Model. Recall the cryptographic timing model introduced by Dwork, Naor and Sahai [DNS04] and explored further [PTV10] in the context of concurrent zero-knowledge (cZK) proofs. There, each participant in a protocol is modeled as a machine that is enhanced with a clock. In the cZK literature, the adversary has full control of the clocks, except that he cannot cause the clocks to go back in time, or cause the clocks of different players to drift more than a parameter τ apart, or cause more than a Δ latency in the transmission of any message.

Our model is inspired by the cZK timing model; however we need to be somewhat stricter, because not only do we need to model the time it takes for protocol messages to be delivered, we also need to model the time it takes for protocol messages to be computed. This is sensitive to the way we model protocol participants; the cZK literature models them as interactive Turing machines, but this is unsuitable if we want to be able to realistically estimate the time it takes them to compute protocol messages. Therefore, also inspired by the work on memory-bound functions [], we model every protocol participant as an interactive RAM machine with a read-only input tape, a write-only output tape, a cache and a memory; its speed of computation is proportional to how many memory accesses it can carry out within a unit of time. Thus, computation that can be carried out using the cache only is essentially free in our model — this can be corrected for, when needed, by simply revising more expensive computational tasks to instruct them to make some number of memory accesses.

Although in our setting there is no adversary, we still assume that the clocks are set up adversarially but within the following constraints:

P_i's clock Each participant P_i has a local clock variable, denoted time_i. The value of time_i can never decrease.

P_i's computation speed Each participant P_i has a computation speed speed_i such that, if P_i is running an algorithm requiring k memory accesses, and $\mathsf{time}_i = t$ at the beginning of the computation of this message, then $\mathsf{time}_i = t + k/\mathsf{speed}_i$ when the message is computed.

τ-synchronization For every pair of participants P_i, P_j, $|\mathsf{time}_i - \mathsf{time}_j| \leq \tau$.

Δ-latency If P_i sends a message to P_j when his local clock is $\mathsf{time}_i = t$, then P_j will have received this message when $\mathsf{time}_i = t + \Delta$.

Following the cZK literature [PTV10], a protocol in this model can make use of these variables as follows: A participant may be instructed to schedule a particular message to go out at a particular time, to measure how much time has elapsed since a particular event happened, and to check if a message from another participant was received by a specific deadline.

Rounds. The timing model naturally yields itself to protocols that proceed in rounds, which the reconstruction protocol does. Let m be the maximum number of steps required for the computation of each round. We assume that the start time of the first round is t_1, which is agreed upon in advance, and the start of each subsequent round $r > 1$ is $t_r = t_{r-1} + \Delta + \tau + m/\mathsf{speed}_{min}$. Every round has the following structure: at local time t_r, each party checks messages on its input tape and determines whether it has enough information to compute s, or whether any other player has deviated from the protocol. If this is the case, it enters the postprocessing phase; otherwise, P_i computes its responses to each P_j, sends out all the responses at local time $t_r + m/\mathsf{speed}_{min}$, and does nothing until time t_{r+1}.

Security, hardness, length and utility parameters. Our protocol uses (Gen, Enc, Dec, Unseal), a time-delayed encryption scheme. Recall that a time-delayed encryption scheme requires a security parameter k and a hardness parameter h. The security parameter k is just the regular security parameter: only strategies whose running time is polynomial in k are considered feasible. The hardness parameter h is set in such a way that even on a machine with speed_{max}, it is impossible to unseal a key in time $\Delta + 2\tau + m/\mathsf{speed}_{min}$ (this is not a circular definition because, as we will see, m here does not depend on h).

Given k and h, let $|\tilde{K}|$ be the length of sealed keys in the time-delayed encryption scheme, and $|c|$ be the length of ciphertexts in the time-delayed encryption scheme. Then we define $\ell = |\tilde{K}| + |c|$. Note that ℓ depends only on k and h. Let (GenVRF, Eval, Prove, Vrfy) and (GenVRF$'$, Eval$'$, Prove$'$, Vrfy$'$) be verifiable random functions (VRFs) with range $\{0,1\}^\ell$ and $\{0,1\}^k$, respectively.

Let $S = \{0,1\}^{|s|}$ be the domain of the secret.

Let β be a parameter that depends on the utilities of the players in the secret sharing protocol; namely, β is such that $U^+\beta + U_i^{guess}(1-\beta) < U$ for every P_i. We will refer to β as the *utility* parameter.

The protocol for n-out-of-n secret sharing runs in time polynomial in k during the sharing and postprocessing phases, and in time polynomial in $h < k$ during the reconstruction phase. The protocol is given in Figure 2.

Sharing phase: Let s be the secret to be shared. The dealer's protocol for the sharing phase is as follows:

1. $K, \tilde{K}, F \leftarrow \text{Gen}(1^k)$
2. Compute $c \leftarrow \text{Enc}_K(s)$, and set $\tilde{s} \leftarrow (c, \tilde{K})$; $\tilde{s}$ is the value that the participants need to reconstruct in the reconstruction phase.
3. Choose $r^* \in \mathbb{N}$, the round in whch $\tilde{s}$ will be reconstructed, according to a geometric distribution with parameter β, i.e. $\forall i \geq 1, \Pr[r^* = i | r^* \geq i] = \beta$.
4. $(pk_1, sk_1), \ldots, (pk_n, sk_n) \leftarrow \text{GenVRF}(1^k)$ and $(pk'_1, sk'_1), \ldots, (pk'_n, sk'_n) \leftarrow \text{GenVRF}'(1^k)$.
5. Choose random $(n-1)$-degree polynomials $G \in \mathbf{F}_{2^\ell}[x]$ and $H \in \mathbf{F}_{2^k}[x]$ such that $G(0) = \tilde{s}$ and $H(0) = 0$. Note that $G(j)$ and $H(j)$ are random ℓ-bit and k-bit strings, respectively.
6. Send sk_i, sk'_i to P_i, and send the following values to all parties:
   - F
   - (pk_j, pk'_j) for $1 \leq j \leq n$
   - $g_j = G(j) \oplus \text{Eval}_{sk_j}(r^*)$ for $1 \leq j \leq n$
   - $h_j = H(j) \oplus \text{Eval}'_{sk'_j}(r^* + 1)$ for $1 \leq j \leq n$

Reconstruction phase: Each player P_i (for $1 \leq i \leq n$) chooses $s_i^{(0)}$ uniformly from $\{0,1\}^\ell$. In each round $r = 1, \ldots$, the players do the following:

1. Except in the first round, P_i checks for messages $y_j^{(r)}, z_j^{(r)}, \pi_j^{(r)}, \pi_j^{(r)}$ from each other player P_j. If P_i has not received these messages from any other player P_j by the beginning of this round, or if for any P_j, $\text{Vrfy}_{pk_j}(r, y_j^{(r)}, \pi_j^{(r)}) = 0$, or $\text{Vrfy}'_{pk'_j}(r, z_j^{(r)}, \pi_j^{(r)'}) = 0$, then P_i aborts and goes to postprocessing. If not, P_i does the following:
   (a) P_i sets $h_j^{(r)} = h_j \oplus z_j^{(r)}$ for $1 \leq j \leq n$, and interpolates a degree-$(n-1)$ polynomial $H^{(r)}$ through the n points $h_1^{(r)}, \ldots, h_n^{(r)}$. If $H^{(r)}(0) = 0$ then player P_i goes to postproccessing immediately after sending its current-iteration message.
   (b) Otherwise, player P_i computes $s_i^{(r)}$ as follows: set $g_j^{(r)} \leftarrow g_j \oplus y_j^{(r)}$ for each other player j. Interpolate a degree-$(n-1)$ polynomial $G^{(r)}$ through the points $g_1^{(r)}, \ldots, g_n^{(r)}$ and set $s_i^{(r)} \leftarrow G^{(r)}(0)$.
2. Each player P_i sends the following to all players:

$$(y_i^{(r)} = \text{Eval}_{sk_i}(r), z_i^{(r)} = \text{Eval}'_{sk'_i}(r), \pi_i^{(r)} = \text{Prove}_{sk_i}(r), \pi_i^{(r)'} = \text{Prove}'_{sk'_i}(r))$$

Postprocessing: Each player i runs postprocessing on the computed $s_i^{(r-1)}$, where r is the round in which player i terminates, as follows:

1. Interpret $s_i^{(r-1)}$ as the tuple $(c_i, \tilde{K}_i)$.
2. Compute $K_i \leftarrow \text{Unseal}_F(\tilde{K}_i)$ and output $\text{Dec}_{K_i}(c_i)$.

Figure 2: Secret Sharing Protocol

3.3 Analysis

The crucial observation for our analysis (following in the footsteps of Fuchsbauer et al., who in turn follow Kol and Naor) is that with all but negligible probability, the set of messages that P_i sends to other players in every round is unique, and P_i cannot undetectably send a different message to any P_j. Thus the only choices P_i faces at every round is simply whether to send out all of his messages (so the resulting strategy yields equivalent play with respect to the original protocol Π), some of them (perhaps probabilistically, for example by sending a message essentially too late for a particular round), or none of them. For the purposes of our analysis, let us discard exact message timing information; all that matters is whether messages arrived on time for a given round.

Suppose all players P_j, $j \neq i$, are following the protocol Π. Suppose that P_i has also been following some strategy that yields equivalent play with respect to Π, up to the beginning of round r. If P_i chooses not to send his round r message to P_j, he will cause P_j to abort (and go to postprocessing) in round $r+1$, so P_i will not hear from P_j from round $r+1$ on. Thus, unless P_i already has sufficient information to compute s by the end of round r, if he chooses not to send one of his round r messages, he will never be able to compute s, and the best he can do is guess.

Let us consider all the information that is available to P_i in round r at the deadline when he needs to either send his round r message or not. There are two cases: (1) P_i already received all the round r messages from all the other players — this corresponds to P_i being similar to a "rushing" adversary in the standard synchronous MPC model; (2) P_i has not received all of the other round r messages.

Suppose we have case (1). Then P_i can compute $s_i(r) = G^{(r)}(0)$ and $H^{(r)}(0)$. By the properties of time-delayed encryption and VRFs, and because P_i does not have time to execute 2^h memory reads in this round, for $r < r^* + 1$ these values are indistinguishable from a random string that is independent of the secret, even with access to auxiliary string aux_i and side information oracle $\mathcal{O}_s^i$. Therefore, computationally, whether r^* is equal to r is independent of his view (an algorithm for P_i that deviates based on whether $r^* = r$ can be used in a straightforward way in a reduction to break the security of either the VRF or the time-delayed encryption), so from P_i's point of view, P_i's expected utility if it continues playing equivalently to Π is at least U (everyone else is following Π, so if P_i does as well, everyone will learn s and get utility U), while if P_i does not send out at least one of its messages, P_i's expected utility is $\beta U^+ + (1-\beta)U_i^{guess} < U$ for our choice of β. Now suppose that $r = r^* + 1$. In that case P_i learns that $H^{(r)}(0) = 0$, and so $s_i(r-1) = \tilde{s}$ is a time-delayed encryption of s. Then even if P_i chooses not to send some of his round r messages, this makes no difference to his utility: everyone will run the postprocessing phase and output s.

Suppose we have case (2). Then P_i has even less information than in case (1), and so the same argument we made in case (1) for $r < r^* + 1$ also applies here. In case $r = r^* + 1$, with the information P_i has it cannot tell whether $H^{(r)}(0) = 0$ (in fact this value is computationally independent of his view), and so his interests are best served by sending out all of his round r messages. Thus, Π is a computational strict Nash equilibrium with side information.

The protocol Π is also stable with respect to trembles. Suppose player P_i believes that, with non-negligible probability δ, players P_{-i} will not follow the protocol (instead following some other strategies, σ'_{-i}). Consider a strategy $\sigma'_i \not\approx_{(AUX,\mathcal{O})} \sigma_i$ for P_i, and consider the case when every other player P_{-i} is following σ_{-i} with probability $(1-\delta)$ and σ'_{-i} with probability δ; let $u_i(\sigma)$ be the utility of P_i from following strategy σ under these conditions. If no player visibly deviates from the protocol Π before round r^*+1, then other players will learn s, and $u_i(\sigma'_i) \leq u_i(\sigma_i) = U$. If σ'_i is such that P_i will not deviate until after another player has done so, then σ_i and σ'_i yield equivalent play with respect to Π, since such a σ'_i will not cause P_i to deviate if no other player does. Therefore, we are only interested in the case when σ'_i deviates first with some non-negligible probability.

Define $p = \Pr[P_i \text{ deviates before any other player }]$. We have $u_i(\sigma'_i) - u_i(\sigma_i) \leq U^+ - U^-$, and this may be achievable when the alternative set of strategies σ'_{-i} are chosen appropriately, with probability δp. With probability $(1-\delta)p$, P_i deviates before any other player, and gets U^+ if he happened to abort in round r^*, and U_i^{guess} otherwise. Finally, with probability $(1-p)$, P_i does not deviate, and gets the same utility whether he was playing σ_i or σ'_i. Putting this all together, we get the following for the expected value of the extra utility P_i gets from following σ'_i:

$$E[u_i(\sigma'_i) - u_i(\sigma_i)] \leq (U^+ - U^-)\delta p + (\beta U^+ + (1-\beta)U_i^{guess} - U)(1-\delta)p$$

Recall that the term $(\beta U^+ + (1-\beta)U_i^{guess} - U)$ is negative (or else P_i is not interested in the publication of the secret). Therefore, there exists $\delta > 0$ such that P_i's expected utility from deviating is negative (for large enough k), even when other players may deviate with some probability δ; this gives us that the protocol is stable with respect to trembles.

We remark that, as proven in [AL09], the protocol cannot be utility independent; that is, a player's utilities for learning the secret must be known in order to properly set β. We also note that the protocol is not resilient to backward induction, since the reconstruction round r^* is less then exponential in k with all but negligible probability (and if it is not, the security of the underlying primitives can be broken).

References

[ABMW05] Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately hard, memory-bound functions. *ACM Trans. Internet Technol.*, 5(2):299–327, 2005.

[ADGH06] Ittai Abraham, Danny Dolev, Rica Gonen, and Joe Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 53–62, New York, NY, USA, 2006. ACM.

[AL09] Gilad Asharov and Yehuda Lindell. Utility dependence in correct and fair rational secret sharing. In *CRYPTO '09: Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, pages 559–576, Berlin, Heidelberg, 2009. Springer-Verlag.

[BCHVS08] Foteini Baldimtsi, Konstantinos Chalkias, Dimitrios Hristu-Varsakelis, and George Stephanides. Mathematical problems and algorithms for timed-release encryption. *Bulletin of the Transilvania University of Brasov*, 15(50):1–4, 2008. Series B.

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *Advances in Cryptology CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Berlin / Heidelberg, 2001.

[CHKO06] Jung Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Timed-release and key-insulated public key encryption. In Giovanni Di Crescenzo and Avi Rubin, editors, *Financial Cryptography and Data Security*, volume 4107 of *Lecture Notes in Computer Science*, pages 191–205. Springer Berlin / Heidelberg, 2006.

[DGN03] Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 426–444. Springer Berlin / Heidelberg, 2003.

[DHR00] Yevgeniy Dodis, Shai Halevi, and Tal Rabin. A cryptographic solution to a game theoretic problem. In Mihir Bellare, editor, *Advances in Cryptology CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 112–130. Springer Berlin / Heidelberg, 2000.

[DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.

[Dod03] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In *PKC '03: Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography*, pages 1–17, London, UK, 2003. Springer-Verlag.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *Public Key Cryptography - PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431. Springer Berlin / Heidelberg, 2005.

[FKN10] Georg Fuchsbauer, Jonathan Katz, and David Naccache. Efficient rational secret sharing in standard communication networks. In Daniele Micciancio, editor, *Theory of Cryptography*, volume 5978 of *Lecture Notes in Computer Science*, pages 419–436. Springer Berlin / Heidelberg, 2010.

[GK06] S. Gordon and Jonathan Katz. Rational secret sharing, revisited. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 229–241. Springer Berlin / Heidelberg, 2006.

[HT04] Joseph Halpern and Vanessa Teague. Rational secret sharing and multiparty computation: extended abstract. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 623–632, New York, NY, USA, 2004. ACM.

[Kat08] Jonathan Katz. Ruminations on defining rational mpc, 2008. Talk given at Summer School on Rational Cryptography, Bertinoro, Italy. Slides available at http://www.daimi.au.dk/ jbn/SSoRC2008/program .

[KN08a] Gillat Kol and Moni Naor. Cryptography and game theory: Designing protocols for exchanging information. In Ran Canetti, editor, *Theory of Cryptography*, volume 4948 of *Lecture Notes in Computer Science*, pages 320–339. Springer Berlin / Heidelberg, 2008.

[KN08b] Gillat Kol and Moni Naor. Games for exchanging information. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 423–432, New York, NY, USA, 2008. ACM.

[LT06] Anna Lysyanskaya and Nikos Triandopoulos. Rationality and adversarial behavior in multi-party computation. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 180–197. Springer Berlin / Heidelberg, 2006.

[MNT09] Peter Miltersen, Jesper Nielsen, and Nikos Triandopoulos. Privacy-enhancing auctions using rational cryptography. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 541–558. Springer Berlin / Heidelberg, 2009.

[MVR99] Silvio Micali, Salil Vadhan, and Michael Rabin. Verifiable random functions. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:120, 1999.

[PTV10] Rafael Pass, Wei-Lung Tseng, and Muthuramakrishnan Venkitasubramaniam. Eye for an eye: Efficient concurrent zero-knowledge in the timing model. In Daniele Micciancio, editor, *Theory of Cryptography*, volume 5978 of *Lecture Notes in Computer Science*, pages 518–534. Springer Berlin / Heidelberg, 2010.

[RSW96] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, 1996.

A Standard Architecture

The following definition of a "standard architecture" is verbatim from [DGN03]. We assume the adversary is limited to a "standard architecture" as follows:

1. There is a large memory, partitioned into m blocks (also called cache lines) of b bits each;
2. The adversary's cache is small compared to the memory. The cache contains at most s (for "space") words; a cache line typically contains a small number (for example, 16) of words;
3. Although the memory is large compared to the cache, we assume that m is still only polynomial in the largest feasible cache size s;
4. Each word contains w bits (commonly, $w = 32$);
5. To access a location in the memory, if a copy is not already in the cache (a cache miss), the contents of the block containing that location must be brought into the cache – a fetch;
6. We charge one unit for each fetch of a memory block. Thus, if two adjacent blocks are brought into cache, we charge two units (there is no discount for proximity at the block level).
7. Computation on data in the cache is essentially free. By not (significantly) charging the adversary for this computation, we are increasing the power of the adversary; this strengthens the lower bound.