PROJECTING COMPUTATIONAL POWER

LEONARD ADLEMAN
UNIVERSITY OF SOUTHERN CALIFORNIA

Abstract. Computational power is not distributed equally among people or organizations. Some people have no computational power, while others control super computers with tens of thousands of processors. There can be little doubt that the major super powers control computational power far in excess of that available to third world countries. In this paper, I will explore how computational power can be used as a bases for novel systems.

1. Introduction

Consider the following scenario:

An American submarine lurks at the bottom of the ocean. Its captain receives the dreaded order to fire nuclear missiles. He proceeds through the complicated protocol designed, among other things, to remove any doubt that the order came from the Pentagon. He fires.

Of course, the real protocol is top secret, but for didactic purposes, lets say it is something like the following:

The Pentagon has a pair of keys $\langle P_P, S_P \rangle$, and the Captain has a pair of keys $\langle P_C, S_C \rangle$:

- The Pentagon signs the fire command, F, with S_P, obtaining A
- The Pentagon encrypts A with P_C, obtaining B
- The captain decrypts B with S_C, obtaining A
- The captain unsigns A with P_P, obtaining F

Is this a good protocol? I suspect it is, but could we do better? Could an adversary with the goal of starting an unintended nuclear war succeed in initiating a launch despite this protocol? Well, that depends. If he knew how to communicate on the appropriate channel, and knew F, S_P, and P_C, then he could.

Notice that the Pentagon would have access to all the information the adversary needs. Further, it seems reasonable to assume, that this would remain the case even for the real protocol.

That information must be protected, but what if its not? Imagine a terrorist obtaining that information by some means. Can we keep him from initiating a launch even in the context of such a security failure?

Date: March 6, 2018.

The answer seems to be that as a practical matter we can.

Here is one way to do it. We add a final level to the existing protocol.

- The captain generates two random primes (of appropriate length) p and q and multiplies them together to obtain n.
- The captain sends n to his correspondent (perhaps the Pentagon, perhaps the adversary) and asks them to return the prime factorization of n.
- The captain fires if and only if he receives the prime factorization of n within 1 second.

Why is this an improvement? The key is the 1 second time limit. What if the primes are chosen of a length that only a super computer could factor n in 1 second? As of 2017, the world's fastest super-computer was the Sunway TaihuLight of China which has 40,960 processors [6] . So, roughly speaking, a number that the Sunway TaihuLight can factor in 1 second, would require about 11 hours on a single processor. Hence by using the protocol, we can ensure that the terrorist cannot initiate a launch unless he has a computer at least as powerful as the Pentagon's.

This is an example of projecting computer power.

2. Knowledge over a channel

What can we learn over a data channel? We have an interlocutor whom we cannot detect except by the messages he sends. He can claim to be anything he wants: beautiful, tall, a Nigerian prince. But, what can he claim that we can confirm with certainty using the channel alone? Our interlocutor can claim to have substantial computational power, and we can make him prove it over the channel by challenging him to compute something fast. In fact, it may be the case, that of all physical attributes, only computational power can be confirmed over a data channel.

Learning over a channel has been considered in the past. For example, in the Turing test of intelligence, an interlocutor is challenged to demonstrate that he can compute like a human, and that he possess the knowledge expected of a human. The more recent CAPTCHA test [7], used to exclude bots from websites, works similarly, but relies more heavily on computation than knowledge.

Cryptographers have searched for systems where the computational cost of decrypting with a secret key is far less than the cost of decrypting without it. Using this approach, they have created practical systems where the combination of secret information and modest computational power can prevail against massive computational power alone.

In this paper we are looking for systems that do not rely on secret information, but where massive computational power can prevail against modest computational power.

Physical power has always played a central role in the physical world; computational power is sure to play as important a role in cyberspace.

3. Power cryptography

Computational power can be used create novel cryptographic systems.

Assume that $\mathfrak{A}$, wants to send a message, M, to $\mathfrak{b}$. There is a adversary, $\mathfrak{c}$, with both read and write access to the channel. Assume that $\mathfrak{A}$ has far more computational power than $\mathfrak{b}$, and far more computational power than $\mathfrak{c}$. Here is how they may proceed.

Assume that there exists a $f : \mathbb{N} \to \mathbb{N}$ such that:

- $\mathfrak{A}$, $\mathfrak{b}$, and $\mathfrak{c}$ can all compute f.
- For all $i \in \mathbb{N}$: $\mathfrak{A}$ can compute $f(i)$ 40960 times faster than $\mathfrak{b}$ and 40960 times faster than $\mathfrak{c}$.
- $\mathfrak{A}$, $\mathfrak{b}$, and $\mathfrak{c}$ all know the standard syntax of messages.
- $\mathfrak{A}$, $\mathfrak{b}$, and $\mathfrak{c}$ all know that the following protocol is to be used.

The protocol:

- $\mathfrak{b}$ chooses an argument i and computes $f(i)$. For convenience, lets say $\mathfrak{b}$ uses $39,600$ seconds (11 hours) for the computation. $\mathfrak{b}$ keeps i and $f(i)$ secret until needed.
- When $\mathfrak{b}$ is informed that $\mathfrak{A}$ wants to send a secret message, $\mathfrak{b}$ sends i along the channel.
- $\mathfrak{A}$ computes $f(i)$ (in less than one second), and encrypts M by computing $C = M \oplus f(i)$ (other cryptographic method may also be used).
- $\mathfrak{A}$ sends C along the channel.
- If within one second $\mathfrak{b}$ receives a string D such that $D \oplus f(i)$ has the standard syntax of a message, then $\mathfrak{b}$ assumes that $D \oplus f(i)$ is a message sent by $\mathfrak{A}$ and reads it. In all other cases, he does nothing.

$\mathfrak{c}$ can also compute $f(i)$, but this will require about 11 hours. In the meantime, the system apparently has the same privacy protection and authentication certainty as the one-time pad. In some settings, such as that described in the submarine scenario, long term protection is not required.

Though the advantage is fleeting, when computational power is used in this way, one acquires some of the advantages of public-key cryptography and some of the advantages of onetime-pads.

Notice also, that this system does not require one-way functions. In fact, its security does not appear to rest upon any unproven complexity theoretic assertions.

4. PITHINESS

Producing messages that attest to computational power has a theoretical basis (see for example [1, 2]). I will not go into great detail here, but will describe the underlying principles informally.

What would you answer if I asked you which of the following is the rarest?

- An atom of lead
- An atom of gold
- An atom of oganesson

Of course, you would answer "what is oganesson?". Oganesson is the highest atomic mass element ever produced by man. It was first synthesized in 2002 by Russian and American physicists using huge amounts of energy. To date, no more than 6 atoms have been detected. Want to change your answer? The correct answer is: oganesson.

Atoms of elements with arbitrarily high atomic mass exist in theory, but most do not exist in the actual universe (let alone on earth). Oganesson does exist and is very rare.

Now consider the following 325 decimal-digit strings. Which is the rarest?

- String-1

  111111111111111111111111111111111111111111111111111111111111111111111111
  11111111111111111111111111111111111111111111111111111111111111111111111
  111111111111111111111111111111111111111111111111111111111111111111111111
  111111111111111111111111111111111111111111111111111111111111111111111111
  111111111111111111111111111111111

- String-2

  117273861574197524263201988378002446322126652202431627149115226334684567
  52692085059991081534315751759185653409176911782072139996207349152935570
  359467605909662847776347920207129957041002218514700426133650898637121515
  341576462485405083282327396750101775882947144815719792547301228279682209
  412331135720349793069947951240428

- String-3

  468172263510722656207776706750069723016189792142528328750689763038394004
  13682313921168154465151768472420980044715745858522803980473207943564433
  527739642811233917558838216073534609312522896254707972010583175760467054
  896492872702786549764052643493511382273226052631979775533936351462037464
  331880467187717179256707148303247

The correct answer is string-3. Why? Because string-3 has a property that very few man made strings ever produced on earth have ever had.

There is a small program (I'll ignore "overhead") that on input of the empty string, output string-1 fast. It is easy to get your hands on as many strings as you want that have this property, so strings with this property are not rare.

The smallest program that outputs string-2 is (with high probability) approximately the size of string-2. String-2 was generated by a random process. It is also easy to get your hands on as many strings as you want that have this property (e.g. by flipping coins), so strings with this property are not rare.[1]

What makes string-3 interesting is that there is a small program that on input of the empty string, outputs string-3, but it is not fast. It is very slow, and there are reasons to suspect that there does not exist a small program that will output string-3 quickly.

String-3 is due to Greg Childers [3]. It is the factorization of $2^{1061} - 1$ (which has two prime factors; string-3 is the concatenation of those factors, smallest first). Childers reports that the factorization took about '3 CPU-centuries' using the Special Number Field Sieve. Other than by factoring $2^{1061} - 1$ (a number which can be generated by a short program), how could you ever come to write string-3 for the first time? I venture to say that since the Special Number Field Sieve is the fastest known algorithm for factoring numbers of this kind, no method available today could have made

[1]Relating strings to the size of programs that produce them has been studied for many decades beginning with Chailtin and Kolomogov.

created string-3 using fewer steps (and hence less energy).

Number theory tells us that the factorization of $2^{1,000,000} - 1$ also exists (and it is not a particularly large), but if it also happens to be the product of two large primes, it only exists in theory, and it is likely that it will never exist on earth and probably never in the universe. If factoring is in fact hard, then there may not be enough energy in the universe to create the prime factorization[2].

Informally, call a string, like string-3, that can be produced by a small program, but cannot be produced quickly by a small program, "pithy". Pithy strings are the rarest things in the universe. The amount of energy needed to produce them grows with their length.

It is easy to define "relative pithiness" by considering program size and running time when the input is not necessarily the empty string. When this is done, it leads us to suspect that if n is the product of two large primes p and q, then p and q are pithy relative to n (however, n is not pithy relative to p and q).

A pithy string can only be made by the application of lots of energy in a sustained process (an algorithm in the broad sense). There is essentially no other way. We can be pretty sure that the human genome is very pithy; being the result of a continuous 3.6 billion year process. Pithy strings can be sent over a data channel; if you receive one, you can be sure that somewhere on the other side of the channel there was a sustained energetic process. When, as in the submarine protocol, we challenge an interlocutor to produce a pithy string of our choosing, we can recognize when he has done so, and by measuring the time between the challenge and the response, we can know that he has expended a considerable amount of energy in a known amount of time. Of course, energy per unit of time is power, and so we can measure computational power over a channel. When our challenge has a time limit, only those with sufficient computational power can respond.

5. remarks on Bitcoin

Bitcoin was introduced in 2008 by Satoshi Nakamoto [4]. At its heart, it relies on pithy strings. A block is acceptable only if when hashed with SHA256 it produces a large number of leading zeros. Hence an acceptable block can be produced with a short program that generates random blocks, applies SHA256, and checks if the result has the required number of leading zeros. That program is very slow (running time exponential in the number of leading zeros), and it is assumed that no short program runs significantly faster.

The Satoshi Nakamoto paper speaks of "proof of work", but it seems more appropriate, both theoretically and empirically, to think of Bitcoin as a system based on "proof of power". "proof of work" and "proof of power" have very different properties.

Mining (and other) rewards are distributed among users in direct proportion to the computational power they bring to bear. As of Aug. 13, 2015, the top ten mining operations were of just two types: those with great computational power acquired by purchasing huge arrays of purpose-built chips, and those with great computational power acquired by organizing huge numbers of users, each with modest computational power; each willing to sacrifice some autonomy. The top three miners accounted for 51% of Bitcoin mining; the top ten accounted for over 90%[5].

[2] If quantum computers become a reality, then this may change, but the arguments given here can also be applied to one-way functions that do not appear vulnerable to quantum speed-up

Bitcoin is largely controlled by the powerful. It is an oligarchy, and there seems little to prevent a state actor, or other agent with great computational power, from making it a monarchy. I suspect that for mathematical reasons this was inevitable (see open problems).

Bitcoin is a very sophisticated system, and an excellent example of the application of computational power. I am quite skeptical about using the current systems for currencies, but I am optimistic that computational power will be the bases for many important applications in the future.

6. Power protocols

In the interest of stimulating further research, I will describe a few more protocols based on computational power. As presented, these protocols are not very practical, but perhaps they will serve as a starting point for others.

6.1. spam filtering protocol.

If I receive an email from someone with whom I have no previous association, there is a non-trivial chance that I will decide not to read it. However, if I notice that the email was sent by Bill Gates, or the President, then I might make an exception. That is, the powerful can get my attention, when those who lack power might not.

So can I automatically filter my email so that I only read such messages when they come from the powerful? Here is one possible protocol.

- When a new email arrives, if it is not from someone in my contact list, then two primes p and q (of appropriate length) are generated at random and multiplied to obtain n.
- An automatic reply is sent that says that the original email will be not be read unless a second email is received within one minute that contains the factorization of n.
- If p and q are received, the original email is placed in the priority box.

The one minute time limit is important here. Without it, I will read the emails received from those who are willing to use energy to get my attention, with it I will read emails from the powerful.

6.2. an election protocol.

On election day 2020, candidate A will face candidate B in an election for President. We will use computational power to vote.

We will assume that a one-way hash functions $H : \{0,1\}^{100} \to \{0,1\}^{20}$ exists with several desirable properties. For example:

- For all $\alpha, \beta \in \{0,1\}^{20}$: The set of preimages of α and the set of preimages of β have the same cardinality.
- For all $\gamma \in \{0,1\}^{20}$: the fastest way to find a preimage of γ is by trial and error. Hence, given γ, the expected number of trials before a preimage is found is 2^{20}.

We will now vote using the following protocol:

- In the first second of election day, the election commission will randomly generate $\alpha, \beta \in \{0,1\}^{20}$. They will post the following instructions on the Internet: To vote for A send a preimage of α, to vote for B send a preimage of β.

- In the last second of election day, the election commission will tally the number, a, of distinct preimages of α they have received, and the number, b, of distinct preimages of β they have received. If $a > b$, A wins; If $a < b$, B wins; If $a = b$, then the protocol will be repeated the following day.

Who wins? With high probability, the candidate for whom the most computational power has been expended. This may seem antithetical to the democratic ideal of "one-man-one-vote"; however, power plays an important role in real election also. In a manner similar to that seen with Bitcoin, the rich invest money to increase their influence, while the poor pool their efforts and organize their work to achieve the same goal.

7. A few open problems

There is much to explore regarding the uses of computational power, here are a few (vaguely stated) problems worth considering:

- Must a system like Bitcoin be based on computational power? Roughly speaking, can a workable, non-altruistic, system (using only data channels) be designed where influence is independent of computational power? I suspect that the answer is "no".
- An analogy: assume that in addition to data, electricity can be sent along the channel. Assume that an interlocutor claims to own a power plant capable of producing 100 watts/sec of electricity. To test him, we can put a 100 watt light bulb at our end and ask him to light it for one second (or any amount of time we would like). Assume he passes that test. Can we conclude that he owns a power plant capable of producing 100 watts/sec? That depends on your definitions. He might own such a power plant, but he might just own a power plant capable of producing 1 watt/hr, and had run that plant for 100 hrs before speaking with you, stored the energy in a battery, and released it to pass the test. When used properly, computational power avoids this kind of uncertainty, since a challenge (for example, the product of two randomly chosen primes) is only generated at the time of the challenge, and so the interlocutor must begin his computation then. Explore this. What does this say about Bitcoin-like systems that substitute money (or other things) for computational power?
- When designing a protocol using computational power, one may choose the amount of pithiness required (for example, in the submarine protocol, by choosing the length of p and q), and the time allowed to produce it. It is the ratio of these two that measures computational power. Of course, many choices give the same ratio. So fix a ratio and lets consider some possible choices. For example, if in the election protocol we had used $H : \{0,1\}^{100} \to \{0,1\}^{80}$, we would have to change the time limit to billions of years in order to get an outcome. This is obviously not a good way to vote. What is the impact of such choices? What is the impact of other choices? For example, what if α and β were release to to the public a year before the election rather than on the day of the election? In Bitcoin, does the time between updates impact the distribution of rewards? Is the time (currently about 10 minutes) between updates merely a method of overcoming the 'noise' produced by network delays?
- Can blockchains be built with a protocol that differs, in some significant way, from that used in Bitcoin?
- Since the power cryptographic system presented above does not require one-way functions, it may be possible to prove some of its properties, unconditionally. This might be the case for other protocols based on the use of computational power. Explore this. What part

does space play in systems based on computational power? Also consider other complexity measures.

- If U possesses a processors, and V possesses b identical processors, investigate protocols that reveal r digits of $\frac{a}{a+b}$. What resources do they use? How much computation (and energy) do they require? How much information must pass?
- The bricks and mortar world has many power structures. For example, typically military organizes have a rank hierarchy, where the most powerful have the highest rank. Sports teams are often linearly ordered on the basis of power. Often these structure are dynamic; if the power of an agent changes (e.g. through injury) his position in the structure changes. Design a dynamic system that ranks participants based on their current computational power.
- Consider n agents, each with some amount of computational power, and design protocols which will allow them to hold a contest and declare a winner. The setting is important. Are the agents allowed to cheat? Is there a central authority? Can the network have delays? Is there a universal clock? Is there a time limit for the contest? What happens if we allow the agents to hold repeated contests *ad infinitum*?

Acknowledgement

The author thanks Ron Rivest for reading an earlier version of this paper and making valuable suggestions.

References

[1] Leonard Adleman, *Time, Space, and Randomness*, MIT Report LCS/TM-131, April 1979.
[2] Charles H. Bennett, *Logical Depth and Physical Complexity*, in The Universal Turing Machine a Half-Century Survey, edited by Rolf Herken, Oxford University Press (1988)
[3] Greg Childers, *Factorization of a 1061-bit number by the Special Number Field Sieve*, MIT Report LCS/TM-131, April 1979.
[4] Satoshi Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, https://bitcoin.org/bitcoin.pdf
[5] Roy Price, http://www.businessinsider.com/bitcoin-pools-miners-ranked-2015-7
[6] https://en.wikipedia.org/wiki/Sunway_TaihuLight
[7] Luis von Ahn, Manuel Blum, Nickolas J. Hopper, John Langford, *CAPTCHA: Using Hard AI Problems for Security*, EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques.