# On the Computational Intractability of Cryptographic Systems

March 15, 2019

## 1 Introduction

Generally, we want to solve problems. In a world where $\mathcal{P} = \mathcal{NP}$, all problems which have reasonably verifiable solutions can be automated. If we had an ability to solve `SAT` efficiently, the scope of our ability to solve problems, both mathematical and practical, would be incredible, and its impact would be huge. Such advancements would dramatically increase efficiency. As Avi Wigderson puts it, a $\mathcal{P} = \mathcal{NP}$ world would be one where jobs, from policing and investigation to economics and engineering, could be coded up. Any mathematical theorem which we could ever hope to be proved would be proven by a computer.

Of course, we conjecture that $\mathcal{P} \neq \mathcal{NP}$. In this world, not only is the above analysis not true, but we lose hope in ever being able to completely automate, in the worst case, a potentially infinite set of problems. The robots may not be all-powerful, but at least mathematicians will still have jobs. In this world, there are problems that are just hard to solve, in the worst case.

Cryptography, the science of protecting information, requires problems which are difficult to solve. Put simply, an encryption scheme relies on parties having some secret key $k$, which can be shared between the two in the case of symmetric encryption or unique to each user in the case of public-key encryption. When encrypting a message $m$, the scheme uses $k$ to create a problem that is very difficult to solve without $k$ but easy to solve with $k$. This description suggests $\mathcal{NP}$.

In an ideal world, we would try to base our scheme off some problem that is in $\mathcal{NP}$ but not in $\mathcal{P}$. Then, we would base our key $k$ off a certificate for this problem and build our decryption algorithm off its verifier. Since our problem is not in $\mathcal{P}$, we've just developed a system which is efficient for users with $k$ but not at all efficient for adversaries without $k$. The attackers are out of luck.

Of course, since we can't prove $\mathcal{P} \neq \mathcal{NP}$, we don't know if we can do that. But we strongly believe it is so. However, we definitely know $\mathcal{NP}$-complete languages are no more likely to be in $\mathcal{P}$ than any other $\mathcal{NP}$ language. Thus, it seems logical to base cryptography off $\mathcal{NP}$-complete languages, to give us the best chance of being correct. This is not done in practice. Instead, modern cryptography is based off of problems like `Factoring` and `Discrete-Log`, which are not known to be in $\mathcal{P}$ but are in $\mathcal{P} \cap \mathcal{NP}$ and hence probably not $\mathcal{NP}$-complete.

People don't believe these languages are in $\mathcal{P}$, but they could be. Progress on other number-theoretical problems have yielded precedent in finding a polynomial algorithm for problems like `Factoring`, but none is clear under current knowledge and classical computers. However, we can factor in polynomial time on a quantum computer. Knowing whether `Factoring` is in $\mathcal{P}$ for classical computers is still interesting. If it is not, then quantum computers will break the Extended Church Turing Thesis. However, with quantum computers potentially arriving in coming decades, cryptographers will need to find a way around it.

This paper will summarize some of the particularities of the Factoring problem and then cover the difficulties of basing cryptography on $\mathcal{NP}$-complete problems, which has not yet been done. These problems amount to near impossibility results, assuming we are correct about certain hypotheses regarding the polynomial hierarchy. Then, we will cover briefly one avenue of cryptography, Lattice Cryptography, where we may still have room to build a basis in $\mathcal{NP}$-completeness. Our goal is to describe a somewhat broad yet interesting dive into cryptography from the complexity-theory perspective.

## 2 Factoring

The Factoring problem can be defined as follows:

$$\texttt{Factoring} = \{\langle N, \ell, u\rangle | N \text{ has a prime factor } M \text{ such that } \ell \leq M \leq u\}$$

If we could decide the above language in polynomial time, we could factor any $n$ in polynomial time by simple binary search over $n$. We hope that we can't do this, though. RSA encryption and digital signature schemes assume and rely on Factoring's hardness, and easy factoring would break the scchemes entirely. Should a polynomial time algorithm be found, the strength of these cryptographic schemes are worthless.

Factoring is known to be both in $\mathcal{P}$ and $\mathcal{NP}$. With certain assumptions, based on unproven number-theoretic conjectures, it can be solved in time $2^{O(n^{1/3})}$ with the General Number Field Sieve. Interestingly, if the requirement that $M$ be prime is removed, the language becomes $\mathcal{NP}$-complete. [Arora-Barak] This additional complexity is a hint for how hard the prime version may be.

As a mathematical problem, it seems logical to try to reduce Factoring to an instance of another $\mathcal{P}$ language, like Linear Programming. However, reducing Factoring to a linear programming instance creates difficulties. This would entail creating some such objective function:

$$\min (n - xy)^2 + x$$

$$\text{S.T.} \quad 1 < x < N$$

$$1 < y < N$$

The minimum of this problem occurs when $xy = n$ and when $x$ is as low as possible. Because neither value can be 1, $x$ is guaranteed to be a non-trivial factor. Moreover, adding the $(+x)$ term to the objective ensures $x$ is the least such factor. The least factor of any number is prime, so solving this objective would get a prime factor of $n$. However, there are two problems: the function is not quasi-convex, so we do not know how to solve it in polynomial time, and there is no restriction on $x$ and $y$ being integers, and it's not immediately clear how to add that restriction in. To see how far we may be from $\mathcal{P}$, notice that if we remove the $(+x)$ from our objective, hence dropping our requirement that $x$ be prime, and add in the requirement that $\ell \leq x \leq u$, we have changed our formula into that $\mathcal{NP}$-complete problem from above.

Sure, this is not a proof that Factoring is not in $\mathcal{P}$. It could very well be, but this suggests that we are dealing with a problem that is close to $\mathcal{NP}$-completeness and cannot be reduced by modern methods.

Discrete-Log has similar issues. We cannot model it as a quasi-convex or convex problem. And years of programming has not generated a polynomial time algorithm. But quantum computers wreck it all. Shor's algorithm

can solve both `Factoring` and `Discrete-Log` in polynomial time. This comes because quantum computers are able to calculate modular exponentation and Fourier transforms exponentially faster than classical computers, allowing them to calculate periods quickly. From here comes `Factoring` and `Discrete-Log`.

# 3 $\mathcal{NP}$ problems and Cryptography

So we want a hard cryptographic scheme. This means it can't be in $\mathcal{P}$. But this may not be enough, with quantum computers bearing down over the extended Church-Turing Thesis, which already can break `Factoring` and `Discrete-Log`, which may not be in $\mathcal{P}$. It would make the most sense, then, to move to $\mathcal{NP}$-complete languages. Quantum computers probably won't be able to solve these problems at all, so it's a good place to look into. People have tried to base schemes on these problems, like the knapsack problem, but failed. Adi Shamir showed that basing cryptography off the knapsack $\mathcal{NP}$-complete problem is less secure than hoped and is often subject to cryptanalysis which can break security.

A quick run through of the terms we will be using to describe cryptography will be useful. We'll be trying to encrypt message out of a certain message space, $\mathcal{M}$. Generally, this space is bitstrings of arbitrary length. To do this, users select keys from a keyspace $\mathcal{K}$ at random. There exists then, generally, two functions $E$ and $D$ which take the key and message as input and encrypt and decrypt any message. Importantly, we must see $D$ inverts $E$, so we can get the message back. We get security if the output of $E$ looks random.[1]

Brassard did work in showing the difficulty of using $\mathcal{NP}$-complete languages in cryptography. He showed a near impossibility result for certain encryption schemes. His theorem is as follows:

**Theorem 3.1** *Consider a public-key encryption scheme with a deterministic encryption algorithm and suppose that the set of valid public-keys is in co$\mathcal{NP}$. Then if retrieving the plaintext from the (ciphertext, public-key) pair is $\mathcal{NP}$-hard then $\mathcal{NP} = co\mathcal{NP}$.*

Public-key encryption is a unique and powerful cryptographic tool, but its specifics are not that important for our discussion. This theorem flows out of a specific definition of a Langauge of 'cryptographic tuples'. Consider a one-to-one Encryption function $E$ over the message space $\mathcal{M}$, with the following property: Given an encryption $c$ of some message $m$, we can determine the size of the original message. Under current schemes, this is a very common property. We can define the following language:

$$L = \{ \langle |x|,\ k,\ c,\ y \rangle \mid x \in \mathcal{M} \text{ and } c = E(k, x) \text{ and } y \text{ lexcographically precedes } x \}$$

If we could efficiently decide $L$, we can easily decrypt any $c := E(x)$ by simple binary search over all $y$. So $L$ is as hard as decrypting $E$. So, if decrypting $E$ were to be as hard as a $\mathcal{NP}$-complete problem, $L$ would also be $\mathcal{NP}$-complete. However, $L$ is also in co-$\mathcal{NP}$.

Consider some tuple $\langle |x|,\ k,\ c,\ y \rangle \notin L$. Since $E$ is a one-to-one encryption function, there is only one $x \in M$ that encrypts to $c$ under $k$, so we just give $x$. If $x$ precedes $y$, we see this tuple cannot be in $L$. Thus, we see $L$ is in co-$\mathcal{NP}$. Since $L$ is $\mathcal{NP}$-complete, this shows $\mathcal{NP} =$ co-$\mathcal{NP}$.

[1] We would like no other polynomial-time algorithm to be able to determine whether the encryptions are truly random bitstrings or outputs from the encryption.

This theorem does provide a degree of impossibility for basing cryptography on $\mathcal{NP}$-complete problems. However, Goldreich and Goldwasser describe schemes that do not satisfy the assumptions made in the theorem, namely that the encryption algorithm is deterministic and that valid keys are in co-$\mathcal{NP}$. There are many probabilistic models. However, still there are problems with basing these models off our hardest problems in $\mathcal{NP}$.

At a more foundational level, One Way Functions (OWF) also pose significant hurdles when trying to base them off $\mathcal{NP}$-complete problems. Simply put, a OWF $f : X \to Y$ is a function that is hard to invert. That is, for any $x \in X$, it is hard to calculate $x$ given $y := f(x)$. It should be clear why they could be so useful in a cryptographic system, where we often want to emit public outputs from which it is very difficult to learn the inputs. In practice, one-way-functions are based off difficult computational problems, like `Discrete-Log` and `Factoring`, or off encryption schemes, yet their use extends to Random Number Generators, Hash functions, and more.

If we try to base an OWF $f$ on $\mathcal{NP}$-complete problems, we run into issues. First, though, we need to clear up what such a $\mathcal{NP}$-complete OWF would look like. Specifically, we show that the task of inverting $f$ is '$\mathcal{NP}$-complete'. Of course, inverting a function is not a decision problem, so we'll build a similar, yet distinct definition for showing the difficulty of inverting $f$. There is a specific type of reduction for this problem, and if we can reduce a $\mathcal{NP}$-complete language to inverting $f$ using this reduction, we'll say inverting $f$ is $\mathcal{NP}$-hard.

This reduction, built by Akavia, Goldreich, Goldwasser, and Moshkovitz, will be based off Oracles. For our function $f$, we'll define the oracle $\mathcal{O}_f$, which can invert $f$ in unit time.[2] We'll say a language $L$ reduces to inverting $f$ if we can build a polynomial time machine with access to $\mathcal{O}_f$, which we'll call $M^{\mathcal{O}_f}$ which decides $L$.[3]

There are two types of these reduction: *adaptive* reductions and *non-adaptive* reductions. First, notice that since $M^{\mathcal{O}_f}$ is poly-time, it can only make polynomially many oracle queries. An adaptive reduction allows $M^{\mathcal{O}_f}$ to base future queries off the results of earlier queries. A non-adaptive reduction does not allow this and forces $M^{\mathcal{O}_f}$ to ask all the queries at once. Clearly an adaptive machine is stronger than a non-adaptive machine.

Akavia, Goldreich, Goldwasser, and Moshkovitz showed this theorem:

**Theorem 3.2** *If, for some OWF $f : X \to Y$ and $y \in Y$ one can efficiently calculate $|f^{-1}(y)|$, the number of elements in $X$ which map to $y$, then, if inverting $f$ is $\mathcal{NP}$-complete under randomized reductions, we see co-$\mathcal{NP} \subseteq \mathcal{AM}$.*

Firstly, we will explain why co-$\mathcal{NP} \subseteq \mathcal{AM}$ is not likely to be true. If we believe $\mathcal{AM} = \mathcal{NP}$, which we do, then necessarily we must believe co-$\mathcal{NP} \subseteq \mathcal{AM}$ implies co-$\mathcal{NP} \subseteq \mathcal{NP}$, which, of course, we do not believe.

To get into the proof itself, consider some OWF $f$ and a $\mathcal{NP}$-complete language $L$ which reduces to $f$. Imagine we can efficiently calculate the number of pre-images under $f$ of some $y \in Y$. We will assume, here, that the number of pre-images is polynomially bounded.[4] We can build an $\mathcal{AM}$ protocol for $\overline{L}$.

Recall that by our definition of the reduction above, we must have a machine $M^{\mathcal{O}_f}$ that given the oracle $\mathcal{O}_f$ can decide $L$ in polynomial time. Now will just build our $\mathcal{AM}$ protocol such that instead of querying $\mathcal{O}_f$, our new machine $M'$ will query Merlin. If Merlin's honest, this works just like an oracle, and $M'$ will decide $L$ and thus $\overline{L}$, but Merlin may not be honest.

We restrict Merlin by requiring that he send back the *entire* set of inversions of some $y$. That is, for the $i$th query

[2] The original scheme allows these oracles to invert $f$ incorrectly with small probability

[3] The original paper uses a $\mathcal{BPP}$-esque definition of 'decides' and just asserts $M^{\mathcal{O}_f}$ is right with probability greater than 2/3

[4] Akavia et al. do not make this assumption, and the analysis can be extended to exponentially large sets of preimages using a Hashing technique similar to showing an algorithm for approximate `#3SAT` in $\mathcal{AM}$ and only looking at an exponentially small subset of the preimages.

$M'$ makes – to invert some $y_i \in Y$ – Merlin returns $A_i = \{x | f(x) = y_i\}$. Notice, we can check any individual $x \in A_i$ to make sure $f(x) = y_i$. Since we can calculate the number of pre-images of $y_i$, we can check if $|A_i|$ is correct. If these two checks are correct, then Merlin has not lied when inverting $y_i$. Merlin must now be honest, and we can use $M'$ to decide $L$ and $\overline{L}$. Thus, we see $\overline{L} \in \mathcal{AM}$. Since $L$ was $\mathcal{NP}$-complete by assumption, the result follows that co-$\mathcal{NP} \subseteq \mathcal{AM}$.

So, if you want to be able to figure out how many elements map to some $y$, you won't be able to build an OWF from a $\mathcal{NP}$-complete problem. This is important. For many implementations of OWFs – like any encryption scheme or `Discrete-Log` or the famous RSA OWF – often have one-to-one OWFs, and so similar OWFs have no hope.[5] We'd need a model where, given some $y \in Y$, the number of preimages of $y$ is essentially random.

Even assuming you can't calculate the number of preimages of $y$ efficiently, we still have problems. Akavia et al. also showed the following theorem:

**Theorem 3.3** *If a non-adaptive reduction exists from* `SAT` *to inverting* $f$*, then co-*$\mathcal{NP} \subseteq \mathcal{AM}$ *follows.*

Given some $\mathcal{NP}$ problem $L$ which has a *non-adaptive* reduction to inverting $f$, we can build an $\mathcal{AM}$ protocol for deciding $\overline{L}$. This construction is difficult to analyze here but builds upon the above construction by bounding the number of preimages from above and below and designing an AM protocol with that bound.

So, it seems that non-adaptive reductions are definitely a no-go when it comes to getting an OWF to be $\mathcal{NP}$-complete. So, many possible schemes are out of luck. We can't have deterministic encryption schemes. We can't have OWFs where you can calculate the number of preimages easily. We can't even have more complicated OWFs, if they can be proven $\mathcal{NP}$-complete under non-adaptive reductions. We'll have to be very creative in our schemes. Again, though, previous work has shown that if we're not careful enough, we'll have problems with cryptanalysis.

Moreover, we need to deal with the distinction between Average-Case complexity and Worst-Case Complexity. $\mathcal{NP}$-completeness is based off worst-case complexity, but most cases may be easy. However, crytographers would prefer an adversary to have only an exponentially small chance at breaking the scheme, and, if more than an exponentially small number of problem instances are not hard, this requirement would be broken. Maybe we could procedurally generate hard instances, but this may limit the number of instances we could have and increase overhead significantly. We'd rather prefer to generate random instances and assume, with high probability, that that instance is difficult.

Bogdinav and Trevisan spend much time discussing the worst-case/average-case trade off in cryptography and the difficulties arising from it. Moreover, Goldreich and Goldwasser assume, without argument, that worst-case hardness is not enough. In a world where deriving instances of these problems needs to be done often by every user in the system, we need creating instances to be cheap. Randomness can help, but then we deal with average-case complexity. We can't be satisfied if there is a non-negligible chance a user selects a bad key. We need problems that are hard in the vast majority of cases, and those cases need to be easy to find.

# 4 Lattice-Based Cryptography and the Shortest Vector Problem

We will conclude on Lattice-Based Cryptography, a leading candidate for post-quantum computing. While we do not have the time to motivate the entire systen, we will provide a brief overview. Ajtai discovered a class of problems,

[5] Brassard's Theorem itself proves this, becuse one-to-one functions can not be hard to invert, and many schemes are one-to-one.

based around Lattices, which are not believed to be efficiently solvable either by classical or quantum computers. We'll quickly define some of the relevant terms. Let $a_1, \ldots, a_n$ be $n$ linearly independent $n$-dimensional vectors. We can define a lattice $\mathcal{L}$ as the set of all linear transformations of these vectors with integer coefficeints. That is, we can define:

$$\mathcal{L}(a_1, \ldots, a_n) = \{\sum_{i=1}^{n} c_i \cdot a_n |\ c_i \in \mathbb{Z}\}$$

The primary problem we will focus on is the Shortest Vector Problem (SVP). SVP asks to find the shortest nonzero vector in some lattice $\mathcal{L}$. That is, we look for $\min_{x \in \mathcal{L}} ||x||$. This objective is not convex. Lovász showed that we can approximate it up to an exponential factor, and Ajtai, Kumar and Sivakumar showed an algorithm to solve it completely in exponential time.

However, Micciancio showed that solving SVP to a constant factor is $\mathcal{NP}$-hard under reasonable assumptions and more-generally, a polynomial time algorithm would force the polynomial hierarchy to collapse to the second level. There are fewer impossibility results for approximating the shortest vector to within a polynomial factor.

Ajtai discovered a way to randomly generate an $n$ dimensional lattice $\mathcal{L}$ and a short vector so that the following holds: if you could build an algorithm to find a short vector in $\mathcal{L}$, you could solve the following 3 problems:

1. Approximate the length of the shortest nonzero vector up to a polynomial factor.
2. Find the shortest nonzero vector $v$ such that any vector with length at most $n^c||v||$ is parallel to $v$.
3. Find the smallest basis up to a polynomial factor.

None of these problems are known to be in $\mathcal{P}$, although Aharonov and Regev showed approximating the SVP up to a $\sqrt{n}$ factor is in $\mathcal{NP} \cap$co-$\mathcal{NP}$. However, where the other problems stand with respect to $\mathcal{NP}$-hardness is still unknown. If we could narrow down the scope of the scheme to rely on constant factors, instead of the polynomial factors relied on by Ajtai, we'd base Lattice cryptography on an $\mathcal{NP}$-complete problem.

From this Ajtai was able to construct a version of the `Subset-Sum` problem which he proved to be at least as hard as the above problem. Regev discovered a system using the Learning with Errors Problem, and Micciancio extended the knapsack problem using Ajtai's lattice techniques. Some of these languages are $\mathcal{NP}$-complete in their base forms, and the Learning With Errors Problem is conjectured to be hard. Although these amended languages are not definitely $\mathcal{NP}$-complete, they are able to have the security that previous attempts have failed to maintain. All of this is not known to be breakable by classical computers or quantum computers. As such, even if the problems lie in $\mathcal{NP} \cap$ co-$\mathcal{NP}$, they are more secure than our current schemes.

In this paper we've shown why current cryptographic schemes, under `Factoring` especially, may be hard. However, we've seen why this is not enough and have motivated attempts to model cryptography on $\mathcal{NP}$-complete problems. We've shown challenges that border on impossibility for certain classes of schemes to do this. Finally, we rounded off one area of hope: Lattice Cryptography, which seems to based off problems relatively harder than `Factoring` or `Discrete-Log`. Basing cryptography on $\mathcal{NP}$-completeness is still an open problem, yet cryptographers do not have the privilege of investing their time into solving this problem for its own sake. With quantum computers on the horizon, something harder needs to be found. Perhaps, we've found something in Lattice cryptography, but more work will need to be done.

# 5 References


D. Aharonov and O. Regev "Lattice Problems in NP ∩ coNP", Journal of the ACM, 2005.

M. Ajtai, "Generating hard instances of lattice problems", Proceedings of the twenty-eigth annual ACM symposium on Theory of computing, 1996.

M. Ajtai, "The shortest vector problem in L2 is NP-hard for randomized reductions", Proceedings of teh thirtieth annual ACM symposium on Theory of computing, 1998.

A. Akavia, O. Goldreich, S. Goldwasser, D. Moshkovitz, "On Basing One-Way Functions on NP-Hardness", Proeedings of the thirty-eigth annula ACM symposium on Theory of computing, May 2006.

S. Arora, B. Barak, "Computational Complexity", Cambridge University Press, 2007.

A. Bogdanov, L. Trevisan, "Average-Case Complexity" Foundations and Trends in Theoretical Computer Science: Vol. 2: No.1. June 2006.

G. Brassard, "Relativized Cryptography", IEEE 20th Annual Symposium on Foundations of Computer Science, 1979.

O. Goldreich, S. Goldwasser, "On the Possibility of Basing Cryptography on the assumption that $\mathcal{P} \neq \mathcal{NP}$", Theory of Cryptography Library, 1998.

D. Lipton, "How Not To Prove Integer Factoring Is In P", Gödel's Lost Letter and P=NP, September, 2012.

L. Lovász, "An Algorithmic Theory of Numbers, Graphs and Convexity," CBMS-NSF Regional Conference Series in Applied Mathematics, 1986.

D. Micciancio, "The Shortest Vector in a Lattice is Hard to Approximate to within Some Constant", SIAM Journal on Computing, 2001.

O. Regev, "On lattices, learning with errors, random lienar codes, and cryptography", Proceedings of teh thirty-seventh annual ACM symposium on Theory of computing, 2005.

A. Wigderson, *Mathematics and Computation*, Princeton University Press, 2018.