CRYPTOGRAPHY IN AN UNCONVENTIONAL MODEL

EDWARD RUGGERI

Abstract. This paper investigates the existence of cryptographic functions and protocols in a model of computation based on compass-and-straightedge (C&S) geometry. The inability to share a secret in this model gives us insight into the nature of both the C&S and traditional Turing Machine (TM) models.

We provide substantial introduction for the benefit of mathematicians not particularly familiar with computability, computational complexity, or cryptography.

1. Cryptography Basics

We begin with a brief review of some fundamental concepts in cryptography. We use numerous examples, motivated by the belief that it is foolish to prove more abstract claims (as we eventually will) without providing first a concrete foundation for students unfamiliar with cryptography. We will make reference to Turing Machines in this first section, but the reader need only know (for now) that they are a model of what a computer can do, and that he may safely ignore the details.

The goal of cryptography is to find (or prove to be nonexistent) constructions that possess primitive cryptographic properties. These are generally related to secure communication. Security properties include, for instance, encryption (message unrecoverable by unintended recipient) and signature (message sender verifiable).

Definition 1.1. A *secret* is a randomly selected quantity, or the value of a function which may only be feasibly computed if in possession of some other secret.

A secret is something which not just anyone should be able to figure out. In particular, deterministic processes (even unfamiliar ones) can always (ostensibly) be replicated by another party. Our definition reflects a good assumption in real life – we develop a more robust model when assuming that one cannot defeat an adversary by using an obscure, but deterministic, method. With regard to theory, it also makes the study of cryptography fascinating by premising security upon (computational) complexity theory.

Definition 1.2. A function f is a *one-way function* (OWF) if f is feasible to compute but any inverse, f^{-1}, is infeasible to compute.[1]

In the usual Turing Machine (TM) model of computation, this means that there is a Turing Machine A which computes f in (*average-case*)[2] polynomial time, but no (average-case) polynomial-time Turing Machine B computes any f^{-1}.

[1]When I write f^{-1}, I will ignore the fact that f may not be injective. Therefore, I will only require that $f^{-1}(y)$ be an element of the preimage of y. I will refer to any such function as an inverse of f. Likewise, I will always treat f as a surjective function.

[2]In cryptography, functions are considered to be feasible/infeasible in the average-case sense, rather than the worst-case sense familiar to complexity theorists.

Remark 1.3. The existence of a one-way function (in the standard model) would imply that $\mathcal{P} \neq \mathcal{NP}$. Put another way, there are some problems which no TM can solve efficiently (not in $\mathcal{P}$) but the solutions of which some TM can verify efficiently (in $\mathcal{NP}$). Due to the use of average-case hardness, it is unknown whether the converse is true.

In particular, any $\mathcal{NP}$-complete problem, by virtue of being as hard as every other $\mathcal{NP}$ problem, would not be solvable efficiently.

Throughout this section, all our examples will be in the standard Turing Machine model.

Conjecture 1.4. *One-way functions exist in the standard TM model.* $\mathcal{P} \neq \mathcal{NP}$.

Example 1.5. A commonly conjectured one-way function is multiplication of two primes ($f(p_1, p_2) = p_1 p_2$). That is, integer factorization of the product of two primes is assumed to be infeasible.

We next define a trapdoor function, but let us first motivate the definition. There are two important qualities of a one-way function: (1) no one can feasibly compute a preimage of $y = f(x)$ given y alone, but (2) anyone can compute f without having to be in possession of a secret, because the security property (non-invertible) is itself not dependent on any secret. This second quality is convenient, because it may be in any case impossible to share a secret securely in some models.

While the first property is very useful in many applications, it is often convenient if some authorized party could recover the input from $y = f(x)$. For instance, anyone may drop a letter in a post box, but the mailman has a key so as to collect them. In this case, the key's shape is the secret which allows the mailman to uniquely open the box. The post office's knowledge of the proper key to use is due to its own selection of the lock.

Analogously, we would like if there were information (let us call it I) which could help an authorized party to invert f easily. However, I cannot be feasibly computed from f, as f is one-way. Therefore, I must be computed from information pre-existing the selection of f – in particular, it is equivalent to information about about how f was selected. Moreover, if I hopes to be secret, it must be computed from another secret quantity. This leads us to the following definition:

Definition 1.6. Let $F = \{f_i\}$ be a family of one-way functions, indexed by a set I, such that:

(1) There exists feasibly computable g such that $g(i)$ describes a procedure to feasibly compute f_i.
(2) There exists feasibly computable h such that $h(i)$ describes a procedure to feasibly compute f_i^{-1}.
(3) There exists a feasible method to arbitrarily sample $i \in I$.[3]

In the standard TM model, the description of a procedure is generally the encoding for a Turing Machine (much like a stored computer program). Assuming that g, h are not secrets, i must not be feasibly recoverable from f_i as then any party could easily compute a procedure to compute f_i^{-1}.

[3]Actually, for cryptographic purposes it suffices to choose in a way which is cryptographically-secure pseudorandom; that is, such that it is infeasible to predict i with greater than negligible probability.

We call F a *trapdoor function family*. When a privileged party randomly selects $i \in I$, then f_i is called a *trapdoor function*. If i is secret, this party alone may feasibly invert f_i.

Conjecture 1.7. *Trapdoor one-way functions exist in the standard TM model. It is unknown whether this is independent of our conjecture that one-way functions exist in the TM model.*

Example 1.8 (RSA Family)**.** First, let us define *Euler's totient function*, the number of coprime factors less than n:

$$\phi(p_1^{k_1} \cdots p_n^{k_n}) = (p_1 - 1)p_1^{k_1 - 1} \cdots (p_n - 1)p_n^{k_n - 1}.$$

Then, let

$$F = \{f_{p,q,e}(x) = x^e \bmod pq : p, q \text{ prime}, 1 < e < \phi(pq), \text{ and } e, \phi(pq) \text{ coprime}\}.$$

That $f_{p,q,e}$ is one-way depends on two assumptions. First, it depends on the difficulty of the *RSA problem*, that finding e^{th} roots modulo a composite N whose factors are unknown is hard. This in turn depends on the assumption that factoring N, the product of two primes, is hard. These assumptions (in the standard TM model) are supported both by convincing (if not conclusive) mathematical argument, as well as the fact that no one has been able to solve the RSA problem efficiently (as far as the public knows).

Now, given the factorization of pq, we easily calculate $d = e^{-1} \bmod \phi(pq)$ (an adversary cannot, as he cannot calculate $\phi(pq)$ efficiently without p, q).[4] We may use d to invert $f_{p,q,e}$ in the following manner:

$$f_{p,q,e}^{-1}(y) = y^d \bmod pq \equiv (x^e \bmod pq)^d \bmod pq \equiv x^{ed} \bmod pq,$$

as groups (in particular, $\mathbb{Z}/n\mathbb{Z}$) are power associative. But $ed \equiv 1 \bmod (p-1)(q-1)$ implies $ed \equiv 1 \bmod (p-1)$ and $ed \equiv 1 \bmod (q-1)$. So, by Fermat's little theorem, $x^{ed} \equiv 1 \bmod p$ and $x^{ed} \equiv 1 \bmod q$. As p, q are distinct, the Chinese remainder theorem implies $x^{ed} = x \bmod pq$.

So f is easily invertible if p, q are known.

Finally, we note that random selection of p, q, e is easy, and these indices allow easy calculation of the one-way function $f_{p,q,e}$ and its inverse.

We now use the definitions of one-way and trapdoor functions to define functions with important cryptographic security properties.

Definitions 1.9. An *encryption function* $f : X \times K \to Y$ is a feasibly computable function which takes an input x (*plaintext message*) and a *key* k and computes a *ciphertext* $y = f(x, k)$. The recipient uses a *decryption function* $g : Y \times K \to X$ (again feasibly computable) to recover $x = g(f(x, k), k')$.

Meanwhile, an unintended recipient who possesses f, g, and y may not feasibly recover x.

If knowledge of k is sufficient for feasible decryption of $y = f(x, k)$, then this is called a *private-key (or symmetric) cryptosystem*. If, in addition to f, g, and y, knowledge of k does not allow feasible decryption of $f(x, k)$, this is called a *public-key (or asymmetric) cryptosystem* (thus $k' \neq k$).

The set K often is referred to as the *keyspace*.

[4]Note the necessity of $e, \phi(pq)$ coprime for d to exist.

The advantage of public-key cryptosystems is that an encryption key can be published for each recipient, and anyone may use this key to send the recipient a message, who will decrypt it with his secret decryption key. This means there are only $2n$ keys in an n-party system (and only n secret keys), rather than $n(n-1)/2$ secret keys. In practice this is a significant reduction in key-management complexity. The downside is that public-key encryption is generally significantly slower than private-key encryption, and relies on the conjectured existence of trapdoor one-way functions.[5]

Example 1.10. The *one-time pad* is a private-key cryptosystem which consists of an encryption function $f(p,k) = p$ **xor** k, and decryption function $g = f$, with a keyspace of bit-strings with length equal to the plaintext.

Note that the *one-time* pad maintains *perfect secrecy* – the probability distribution over the message space M is equal to the probability distribution over M conditional on any one ciphertext $c = m$ **xor** k. Thus, if the key is selected randomly, then the one-time is unbreakable in the sense that it is not only infeasible, but in fact impossible for an adversary to learn any additional information about the message given the ciphertext.

The name *one-time* pad emphasizes the necessity in using a key only a single time. Otherwise, one can guess k by taking the intersection of the keys leading to possibly valid decryptions of c_1 and the keys leading to possibly valid decryptions of c_2.

Example 1.11. The *RSA cryptosystem* is a public-key cryptosystem based on the RSA trapdoor function family. The private and public keys are e, d (as defined earlier) respectively. The encryption and decryption functions are $f_{p,q,e}, f_{p,q,e}^{-1}$ respectively.

Definitions 1.12. A *protocol* A is a series of private computations and public communications between a set of parties (in general, two).

A *zero-knowledge proof of knowledge* A about a secret x is a protocol between two parties (the *prover* and the *verifier*) such that upon completion of A:

(1) An honest verifier will be convinced that an honest prover knows the value of x.[6]
(2) No dishonest prover can convince an honest verifier that he knows the value of x.
(3) The (honest or dishonest) verifier does not learn anything about the value of x.

The following zero-knowledge proof relates to graph theory. A *Hamiltonian cycle* is a cycle which visits each vertex of a graph using no edge more than once, a graph is *Hamiltonian* if it contains such a cycle. The $\mathcal{NP}$-complete *Hamiltonian cycle problem* is to find a Hamiltonian cycle in a given Hamiltonian and is, by our conjecture, infeasible to solve. The *graph isomorphism problem* is another $\mathcal{NP}$-complete problem which asks us to determine whether two graphs are isomorphic.

[5]In the TM model, one-way (not necessarily trapdoor) functions are required for private-key systems to withstand certain attacks from adversaries with slightly greater, but still plausible, abilities. However, that is beyond the scope of our current discussion.

[6]The standard is not necessarily that of mathematical proof – the probability that the verifier is fooled need only be less than some ϵ (*soundness error*) considered to be sufficiently small.

Example 1.13. Say Alice knows a Hamiltonian cycle in a graph G. She wishes to prove this to Bob, without revealing if any particular edge is in the cycle. To do this, they play several rounds of a two-step game:

(1) Alice creates a graph H isomorphic to G which she shares with Bob. Trivially, she knows a Hamiltonian cycle in H.
(2) Bob flips a coin to decide whether to ask for the isomorphism between G and H (which he verifies), or he can ask for the Hamiltonian cycle in H (which he verifies).

An impostor of Alice must be ready to provide both a valid isomorphism between G and H and a valid Hamiltonian cycle in H, since Bob might ask for either of these. If the impostor were able to do this, though, then he would know a Hamiltonian cycle in G, which violates our conjecture that he cannot solve an $\mathcal{NP}$-complete problem (Hamiltonian cycle problem). Thus an impostor will fail one run of this game with probability at least $\frac{1}{2}$. After n runs, the probability that an impostor has not yet failed a run is at most $\frac{1}{2^n}$.

Bob, on the other hand, learns nothing about the Hamiltonian path in G. An isomorphic graph H cannot help him calculate a Hamiltonian path in G – they are the same graph, up to a relabeling of edges and vertices. On the other hand, if Bob could use the Hamiltonian path in H to discover one in G, he would also have found an isomorphism between G and H. But because the graph isomorphism problem is $\mathcal{NP}$-complete, he cannot do this either.

Example 1.14 (Diffie-Hellman Key Exchange)**.** This fascinating protocol is used by a pair of parties to generate and share a secret over an insecure channel.

(1) Alice and Bob (publicly) agree on a prime number p and an integer g.
(2) Alice (privately) chooses an integer a, Bob (privately) chooses an integer b. They exchange $a' = g^a \bmod p$ and $b' = g^b \bmod p$.
(3) Alice computes $(b')^a \bmod p$ and Bob computes $(a')^b \bmod p$. Since groups are power associative, Alice and Bob have both calculated the same number.

The inability to compute the secret even given the public communication between Bob and Alice is premised on the *Diffie-Hellman problem* (DHP): Given g in a cyclic group G and g^x, g^y, calculate g^{xy}. The difficulty of DHP is unknown: currently the most efficient solution is to solve the *discrete log problem* (DLP) – find x from g^x – which is assumed to be hard. Furthermore, significant progress has been made toward showing that DHP is indeed as hard as DLP (in all but the most trivial cases) and that an efficient solution to DHP would imply the existence of an efficient solution to DLP, which is seen as highly unlikely.

2. The Turing Machine Model of Computation

A *model of computation* consists of rules which define what kind of computation can be performed feasibly, if at all. From the previous section, our definitions have been carefully selected so as to be independent of any particular model (though all our examples were from the standard TM model). However, what kind of cryptography is possible, as well as its nature, can be very different from model to model.

The traditional, standard model is that of the Turing Machine. We have already relied on the most basic familiarity with Turing Machines (e.g., that they are a

model of a computer, $\mathcal{P}$, $\mathcal{NP}$), but we give a very brief, very informal definition now. It is not quite the traditional definition (though it is equivalent), as it hopes to be more immediately intuitive.

Definition 2.1. A *Turing Machine* consists of:

(1) Two tapes: each an infinite-length array of cells (but with leftmost cells). Each cell may contain one character of a finite alphabet, which includes a blank symbol. We assume the alphabet is $\{0, 1, B\}$.

    The input is initially written on the input tape, left-justified. The second tape is the output tape – when the machine is done computing, the output is left-justified on this tape. Both tapes may be read from and written to.

(2) Two tape heads: a moving pointer to the cell currently being read from and written to, one for each tape. They both initially point to the leftmost tape cell.

(3) A finite set of machine states: the way the machine behaves depends in part on what state it is in. There is a specified initial state. There is also a halting state: when the machine is done computing, it enters the halting state and stops.

(4) A transition function: The transition function takes as input the machine state and the contents of the cells underneath both tape heads. It decides what to write to the cells (it may write back the same character), how to move (independently) each of the tape heads (left, right, or not at all), and what state to change to (if at all).

    The transition function is repeatedly applied until the halting state is reached. If the halting state is never reached, then the TM fails to terminate and properly complete its output.

    We allow the transition function to be non-deterministic – it may also take as an additional input the result of a random coin flip. This may lead to different computational paths and different outputs on different runs, but on any single run, only one computational path will be followed.[7] Note that on different runs, the same Turing Machine may return different outputs (e.g., a random number generator).

Our definition of a TM already prescribes what kind of computation is possible. For instance, Alan Turing showed in 1936 that no TM can solve the Halting Problem: that it is impossible to design a TM which can decide whether any given TM will halt on any given input. The existence of a TM A which solves the Halting Problem would present a contradiction, since we may then design a TM $B(x)$ which uses A to halt iff $A(x, x)$ returns 1 (that is, halts if the program described by the input of B halts on its own description). But then, if y is a description of B, we ask the value of $A(y, y)$. This returns 1 iff $B(y)$ halts, which halts iff $A(y, y)$ returns 0. So we have reached a contradiction.

But of the computations which are possible, we now make a well motivated, yet somewhat arbitrary decision that only some are *feasible*. There are two primary computational resources: memory (number of tape cells used) and average time (the

[7]This differs somewhat from the common definition of a non-deterministic TM which, in turn, motivates the definition of $\mathcal{NP}$. Those follow *all* computational paths, which may be thought of as spawning additional machines. They are said to terminate as soon as one of the spawned machines terminates.

number of steps) used over all inputs and computational paths. We reduce memory to time, however, by noting that the amount of time a TM takes to compute implies an upper bound on the number of tape cells used.[8]

We say a TM A is feasibly computable if the average-case number of steps required is *polynomial* in the length of the input – that is, there exists a polynomial P such that $P(n)$ is greater than the average number of steps required to compute $A(x)$, where x is a problem instance of length n bits.

The complexity class $\mathcal{P}$ consists of problems which are solvable by Turing Machines in *worst-case* polynomial time. Thus, any problem in $\mathcal{P}$ is feasible. $\mathcal{NP}$ consists of problems which are solvable in worst-case polynomial time by branching, non-deterministic Turing Machines (not the single computational-path kind). Equivalently, $\mathcal{NP}$ consists of problems whose solutions are verifiable in worst-case polynomial time by a standard TM. $\mathcal{NP}$-complete problems are $\mathcal{NP}$ problems which are at least as hard as any other $\mathcal{NP}$ problem. Any $\mathcal{NP}$-complete problem is infeasible by conjecture that one-way functions exist.

Among over virtues, this approach makes feasibility a relative and scalable property. It reflects the important question: As computing power increases, will our ability to solve more complex problem instances keep up with our ability to construct them? In practice, any problem not solvable in average-case polynomial time (such as an $\mathcal{NP}$-complete problem) can be scaled to such a size that it is infeasible to solve it.

3. Nonstandard Models of Computation

In the field of computability – the study of what kind of computation is possible – the Turing Machine model has reigned supreme. The (Alonzo) Church-Turing Thesis states, informally, that Turing Machines represent the limit of computability: any function which one might naturally regard as computable can be computed by some Turing Machine. Though not amenable to mathematical proof or disproof, many computability theorists accept the Church-Turing Thesis, and thus eschew the belief that hyper-computation (computing functions not TM computable) is possible or even reasonable. So far, the thesis is well supported by practice – real life computation is much like that done on theoretical Turing Machines.

On the other hand, the supremacy of Turing Machines in the field of computational complexity is much less assured. Quantum computers (provably) cannot push beyond the computation possible on Turing Machines, but algorithms have already been designed to efficiently solve, on quantum computers, problems which are believed to be infeasible on Turing Machines. For instance, (Peter) Shor's algorithm can factor an integer in polynomial time on a quantum computer, even though this problem is conjectured to lie outside of $\mathcal{P}$. Indeed, the security of the RSA cryptosystem relies on this assumption, so the construction of sufficiently powerful quantum computers would threaten the continued use of this protocol. Even if practical difficulties in constructing quantum computers prohibit their replacement of conventional machines, the theory has already demonstrated that perfectly reasonable, alternative models of computation exist, with the ability to feasibly solve problems considered to be intractable in the traditional TM model.

[8] In real life, memory is frequently much more scarce than time, but this need not concern us for now.

4. The Compass-and-Straightedge Model of Computation

One particularly interesting, nonstandard model of computation dates from the time of the ancient Greeks, but was only recently proposed as a setting for cryptography by Mike Burmester, Ronald Rivest, and Adi Shamir. This is the *compass-and-straightedge* (C&S) model of computation. Computation takes place in the plane, and the only tools available for use are an unmarked, infinite-length straightedge and a compass (in Euclidean geometries, we will show it doesn't matter whether the compass collapses upon being lifted from the page), along with the power to randomly choose points in the plane. Given two points, we may draw the line which passes through them, or the two (possibly indistinct) circles centered at one and passing through the other. New points can be constructed if they lie at an intersection of two non-parallel lines, two distinct circles, or a line and a circle. Any such construction which is possible is considered to be feasible, but any computable procedure in the C&S model must take a finite number of steps.

Remark 4.1. I have purposefully included no diagrams, encouraging the reader to follow along with his or her own straightedge and compass.

Theorem 4.2 (Compass Equivalence Theorem)**.** *Given $C(A, |AB|)$ (a circle centered at A which passes through B), for any A' we may construct $C(A', |AB|)$ using only straightedge and collapsing compass.*

Proof. It suffices to show that given three points, A, B, C, we may produce a point X such that $|AX| = |BC|$. First, construct the line segment $\overline{AB}$. We may then construct an equilateral triangle, $\triangle ADB$, where D lies at the intersection of two constructable circles: $C(A, |AB|), C(B, |BA|)$. Using our straightedge, we may extend $\overline{DB}$ out to a point E, where it intersects the circle $C(B, |BC|)$. We now extend $\overline{DA}$ to a point F, where it intersects $C(D, |DE|)$.

Clearly $|DF| = |DA| + |AF|$, and $|DF| = |DE| = |DB| + |BE|$. But $|DA| = |DB|$ by the construction of the equilateral triangle. So $|AF| = |BE|$, and since $|BE| = |BC|$ by the construction of the circle, $|AF| = |BC|$. □

Remark 4.3. When convenient, I will translate circles (and distances) in the plane without resorting to this construction which justifies my doing so.

It is useful to provide an algebraic description of this model. We may treat the plane as the complex field by selecting one point randomly as the origin $O = 0$ and a second as $e_x = 1$. What complex numbers may we construct from these first two? Clearly not even all of $\mathbb{R}$, since we can only construct at most a countable number of points using finite processes. But first let us introduce some notation.

Notations 4.4. Suppose $S \subset \mathbb{C}$. Define $\gamma(S)$ to be the union of S and the points in the intersection of distinct lines and/or circles. So $\cup_{n\in\mathbb{N}}\gamma^n(S)$ is the closure of S with respect to compass-and-straightedge operations. Define S_0 to be $\{O, e_x\}$. Let $S_n = \gamma^n(S_0)$. We call the compass-and-straightedge closure of S_0 the *constructable* points, and denote it S_∞.

Theorem 4.5. *$z \in S_\infty \subset \mathbb{C}$ iff there exists a finite tower of quadratic field extensions $\mathbb{Q} \subset K_0 \subset \ldots \subset K_n$, $z \in K_n$.*

Lemma 4.6. *S_∞ forms a field.*

More generally, given $0, 1, a, b \in \mathbb{C}$ we may construct:

(1) $a + b$
(2) $a - b$
(3) ab
(4) a/b *(*$b \neq 0$*)*

Proof. We prove each item in turn, switching somewhat freely between geometric and complex notations.

(1) Using the Compass Equivalence Theorem, we may draw $C(B, |OA|)$ and $C(A, |OB|)$. Clearly $a + b$ lies on both of these circles, so it must be in the intersection of the two as well.
(2) It suffices to construct $-b$, which we may do by drawing $C(O, |OB|)$, and finding the point at the opposite end of the diameter. Then we simply add this to a.
(3) First, we demonstrate how to construct segments with lengths a_x, a_y given $a \in \mathbb{C}$. First, construct $C(A, |AO|)$, which intersects the x-axis at O and some other point C. The intersection of $C(O, |OC|), C(C, |CO|)$ is a pair of points which describe the line perpendicular to the x-axis passing through A. We then have a right triangle with hypotenuse length $|OA| = a$, and leg lengths a_x and a_y. We may do the same to obtain line segments with lengths b_x and b_y.

    Now, $ab = (a_x b_x - a_y b_y) + (a_x b_y + a_y b_x)i$. So the next step is to multiply line segment lengths (e.g., construct a segment with length $a_x b_x$). First, we draw the perpendicular to the x-axis at e_x as before. Using the Compass Equivalence Theorem, we may find a point X on this perpendicular such that $|e_x X| = a_x$. This defines a triangle, $\triangle O e_x X$. Now we select a point Y on the x-axis such that $|OY| = b_x$, and we draw a perpendicular. At a point Z, this will intersect an extended $\overline{OX}$ and form $\triangle OYZ \sim \triangle O e_x X$. The segment $\overline{YZ}$ will thus have length $a_x b_x$.

    We may form segments of lengths $a_y b_y, a_x b_y, a_y b_x$, too. We may then add/subtract them to obtain segments with lengths $a_x b_x - a_y b_y$ and $a_x b_y + a_y b_x$. By lying the first on the x-axis and extending the second perpendicular, we construct the point $C = ab$.
(4) $a/b = a\bar{b}/b\bar{b}$. We can construct $\bar{b}$ and thus $a\bar{b}$ and $b\bar{b}$. We may find the inverse of $r = b\bar{b}$ using the same kind of triangle trick as before: take a right triangle with leg lengths $r, 1$ and make a similar triangle with legs $1, 1/r$ (note that we cannot do this if $r = 0$). We may then multiply $a\bar{b}$ by the new quantity $1/r$.

□

Because $0, 1, i \in S_\infty$, $\{r + qi : r, q \in \mathbb{Q}\} \subset S_\infty$. But we now extend this field through closure under square roots.

Lemma 4.7. *S_∞ is closed under square roots. Thus, any z which belongs to the top of a finite tower of quadratic field extensions of $\mathbb{Q}$ is also in S_∞.*

Proof. To construct square roots, it suffices to construct square roots of real numbers. For complex numbers, we may always find their real and imaginary parts as

in the lemma, and then apply the formula:

$$\sqrt{x+iy} = \sqrt{\frac{\sqrt{x^2+y^2}+x}{2}} + i\frac{y}{\sqrt{2(\sqrt{x^2+y^2}+x)}}$$

So given $a \in \mathbb{R}$, construct the point $A' = 1 + a$. We may construct the circles $C(O, |OA'|), C(A', |A'O|)$ and find a perpendicular to the x-axis that divides $\overline{OA'}$ in half at the point of intersection X. We may then draw the circle $C(X, |XO|)$ which clearly passes through both O, A'. Now, we may also construct a perpendicular to the x-axis at e_x. This intersects the circle at two points, choose one and call it Y.

We may now create the similar triangles $\triangle A'e_xY \sim \triangle Ye_xO$ (since $\angle OYA' = \pi/2$, $\angle A'Ye_x = \angle YOe_x$). So,

$$\begin{aligned}
\frac{|A'e_x|}{|Ye_x|} &= \frac{|Ye_x|}{|Oe_x|} \\
\frac{a}{|Ye_x|} &= \frac{|Ye_x|}{1} \\
a &= |Ye_x|^2 \\
|Ye_x| &= \sqrt{a}
\end{aligned}$$

By induction, an element of at the top of any finite tower of quadratic field extensions of $\mathbb{Q}$ is constructable. □

Proof of Theorem 4.5. We now need only show that any constructable point is at the top of a finite tower of quadratic field extensions of $\mathbb{Q}$.

For an induction proof, it suffices to show that any point constructed from a set S must be algebraic of degree at most 2. There are three ways to construct points. If the new point lies at the intersection of two lines, it is the solution of a pair of simultaneous linear equations – it is algebraic of degree 1. If the point lies at the intersection of a line and a circle, we may solve the linear equation for y and substitute the expression into the equation for a circle to obtain a quadratic equation in x. The intersection of two circles may be described by subtracting the equation for one from another, which results in a linear equation when the squared terms cancel. This may then be substituted into either of the equations for the circles, to obtain another quadratic equation.

We have thus proved inclusion in the other direction. S_∞ is exactly those points which lie at the top of finite tower of quadratic field extensions of $\mathbb{Q}$. □

5. A Comparison of Cryptography in Both Models

We have now developed two very different models of what is feasibly computable. Next we investigate what kind of cryptography is possible in the C&S model, and how this compares to the traditional TM model.

First, we note that in both models we have the ability to generate random numbers: rationals in the TM model, complex numbers in the C&S model. Secrets should not come from a countable set in the C&S model, since adversaries can enumerate the set and in general discover the secret.

We have only been able to conjecture the existence of one-way and one-way trapdoor functions in the TM model. In the C&S model, we can demonstrate the existence of a one-way functions.

Example 5.1. Sample a point $z \in \mathbb{C}$. This defines an angle $\angle ZOe_x$. It is easy to construct an angle which is triple this. Further, it is elementary to prove that one cannot trisect this arbitrary angle in the C&S model. So constructing the triple is a one-way function.

Burmester, Rivest, and Shamir used this as the basis of a zero-knowledge proof of knowledge.

Example 5.2. Alice wishes to prove that she can construct the X, the trisection of an arbitrary angle Y. She publishes Y, knowing that no one can construct X from this with probability greater than zero. To prove to Bob that she can trisect Y, they use a two-round protocol:

(1) Alice constructs an arbitrary angle, K, and triples it to obtain R. This she gives to Bob.
(2) Bob flips a coin to either ask for K (which he verifies is the trisection of R) or for $K + X$ (which he verifies is the trisection of $R + Y$).

This is repeated n times, at which point the probability that Alice cannot trisect Y yet has completed all n rounds successfully is 2^{-n}.

Anyone who cannot trisect Y cannot construct both K and $K + X$, for then he could construct X, the trisection of Y. So an impostor fails any one round with probability $1/2$.

On the other hand, Bob gains no knowledge of X. If he requests the trisection of R, certainly he learns nothing about X. Even if he asks for $L = K + X$, he could have simulated this by selecting an angle L at random and solve $3 * L = R + Y$ for R. Thus he learns nothing that he couldn't have computed on his own.

We now show that private encryption schemes exist. In particular, we show that the analogue of a one-time pad is feasibly computable in the C&S model.

Example 5.3. Consider $f : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$, $f(z, k) = z + k$. Then f is a private key encryption function which encrypts a message z using a secret key k. The decryption function is simply $g(z, k) = z - l$.

However, a necessary requirement for using a private encryption function is a shared key. It is conjectured that sharing a secret over an insecure channel in the C&S model is impossible. See, for instance, David Woodruff and Martin van Dijk who prove that in a simplified model of geometric cryptography where parties work over $\mathbb{Q}$ and perform standard field operations, any point constructable by both Alice and Bob is constructable by an observer of the communication between them.

References

[1] M.Burmester, R. Rivest, A.Shamir. Geometric Cryptography. Unpublished. 1997.
[2] U. Feige, A. Fiat, A. Shamir. Zero Knowledge Proofs of Identity. Journal of Cryptography. 1988.
[3] J. Rompel. One-Way Functions are Necessary and Sufficient for Secure Signatures ACM Symposium on Theory of Computing. 1990.
[4] P. Woodruff, Marten van Dijk. Cryptography in an Unbounded Computational Model. Eurocrypt 2002.