Why the Ricardian Contract Came About: A Retrospective Dialogue with Lawyers

IAN GRIGG[*]

Abstract

In the halcyon days of 1995, I needed a way to capture the nature of financial agreement in order to create an instrument that could be traded on the Internet. By way of the zero coupon bond, it was discovered that the legal prose contract is close to the semantic essence of all financial instruments; the way to issue any instrument reduces to the way to capture any contract, digitally and cryptographically. The "Ricardian Contract" emerged as the design pattern for that capture: a human-readable, contractually significant document, digitally signed and including sufficient but simple markup tokens such that a computer program could extract out the handful of important values: face, rates, issuer, etc. The document could then be hashed cryptographically, providing a secure, unique and cost-free identifier. With Bitcoin's drive to decentralise all of finance, the design pattern is now emerging as the way to efficiently tie a legal intent into a financial agreement.

Introduction

How do you issue a financial instrument onto the Internet? How do you securely negotiate a financial agreement over the net, and not end up in court? How do you capture all of the legal significance of a deal in such a way as to reduce confusions and empower traders? The Ricardian Contract is a tool to meet these goals—a method to capture the essence of any deal for Internet trading.[1] This article will show how it came to be, and why it came to take the form it took. It will also show its relationship to the related topic of "smart contracts".

In early 1995 I was sitting in Finance 1 classes at London Business School learning about the marvels of the "zero coupon bond". The "zero" is a promise to pay a sum on a date. An issuer will write a zero coupon bond to raise capital to finance good works, do those works, and pay back slightly

[*] *iang *at* iang *dot* org - An early version of this article was presented at R3's first Smart Contract Templates Summit, London/New York 2016

1 Ian Grigg, "The Ricardian Contract," 2004, IEEE 1st Workshop on Electronic Contracting http://iang.org/papers/ricardian_contract.html

more on the stated date; their peculiarity is that it is hard to get anything simpler as a financial instrument.

This very simplicity gives the zero a special role in finance: we can build a *mathematica financia* or financial language modelled on debt. A dollar due on a given date can be used to compose many advanced financial products. For example, take a zero and make it defaultable: it could pay out, or not pay out by a certain date. Add an event by which it does not pay out, and we now have an *option*. Apply mathematics to the option over the zero and we eventually get Black-Scholes, the formula that predicts a price for the option, and arguably did more than any other factor to drive the modern finance industry in the late 20th century.[2]

The takeaway here is that the zero is the atomic unit of finance. The relevance of this was to be found in Amsterdam, seat of the world's oldest stock exchange, and in the 1990s, the start of a new revolution in finance.

1. The Origins of the "Riccy"

Blinded by cryptography

In Amsterdam, my cypherpunk friend Gary Howland was working with other cryptographers and programmers to make an entirely new and interesting form of the dollar: David Chaum's invention of "digital cash" that could be transmitted across the Internet and arrive safely and *untraceably* at a recipient.[3] In the mid 1990s, this cryptographic money was terribly exciting; the web was only just seeping onto the public consciousness, and already e-commerce was squeaking its first post-natal cries—*where's the money?*

Chaum's eCash was the answer. The money was everywhere, nowhere and in Amsterdam, all at the same time; his formula to preserve monetary privacy was a sensation in an already hyperbolic market inspired by the browser, the World Wide Web and thousands of startups. While watching my professor turn one zero into many—into options, the Ho Lee binomial model, into Black-Scholes—my mind was swirling with the impact of this new Internet dollar.

[2] On the Black-Scholes model, see Manlio Del Giudice, Federica Evangelista and Matteo Palmaccio, "Defining the Black and Scholes approach: a first systematic literature review" (2015) 5 *Journal of Innovation and Entrepreneurship* 5.

[3] David Chaum (1983). "Blind signatures for untraceable payments," Advances in Cryptology Proceedings of Crypto 82 (3): 199–203.

Suddenly, in class, it clicked into place: eCash was more or less a zero coupon bond. There was no explicit payment date, but so what? If we could issue a dollar on the net, we could also issue a zero, and if we could do that, we could use the finance techniques unfolding on the board in front of me to issue practically everything else. This connection brought the excitement of the new Internet, the new eCash to the boring old zero, to finance classes, and to me. If eCash was exciting to geeks and boring to the City, what was exciting in finance? Derivatives! Shares! Commodities! Especially, derivatives could be constructed as composites of the zero coupon bond and complicated formula followed. Efficient market hypothesis! Black-Scholes! Profits!

If I could reduce Chaum's eCash into the tokens of finance known as zeroes, I could also issue all of the financial instruments that humanity had dreamed of. Suddenly, eCash became exciting to financiers as well as geeks. In 1994, DigiCash did not appear to be thinking of finance, but Nick Szabo was:

> *Another area that might be considered in smart contract terms is synthetic assets. These new securities are formed by combining securities (such as bonds) and derivatives (options and futures) in a wide variety of ways. Very complex term structures for payments (i.e., what payments get made when, the rate of interest, etc.) can now be built into standardized contracts and traded with low transaction costs, due to computerized analysis of these complex term structures. Synthetic assets allow us to arbitrage the different term structures desired by different customers, and they allow us to construct contracts that mimic other contracts, minus certain liabilities.*[4]

Research

[4] Nick Szabo, "Smart Contracts," 1994
originally: szabo.best.vwh.net/smart.contracts.html but can be found at: https://web.archive.org/web/20011102030833/http://szabo.best.vwh.net/smart.contracts.html

Gary Howland and I resolved to give it a go—to build a crypto system to issue any financial instrument on the net, and trade it. Like many computer scientists in the 1990s, we embarked on a journey to build an Internet of finance, a long journey that in time became known as "Ricardo" after the British Economist who discovered that free trade is better for everyone.

Gary Howland concentrated on the lower cryptosystem, called SOX, and I concentrated on the higher financial application.[5] My first step was to research all of the financial instruments out there and pick one as a minimum viable product. I chose bonds because I perceived them to be simple, because I had studied them in class, and because the market was easy to enter. Bonds were simpler than other financial instruments. Equity, for example, was harder to model; commodities required an underlying; derivatives included all their own complexity as well as reference to an underlying. The simplicity of the bond's certain payouts was attractive. Bonds were also lightly regulated in comparison to some other types of security, making for a market that seemed to have low barriers to entry - important for a small fintech startup. Either way, rightly or wrongly, at the time I perceived bonds to be a market that was easy to enter.

What complexity did exist in bonds was found in two parts:

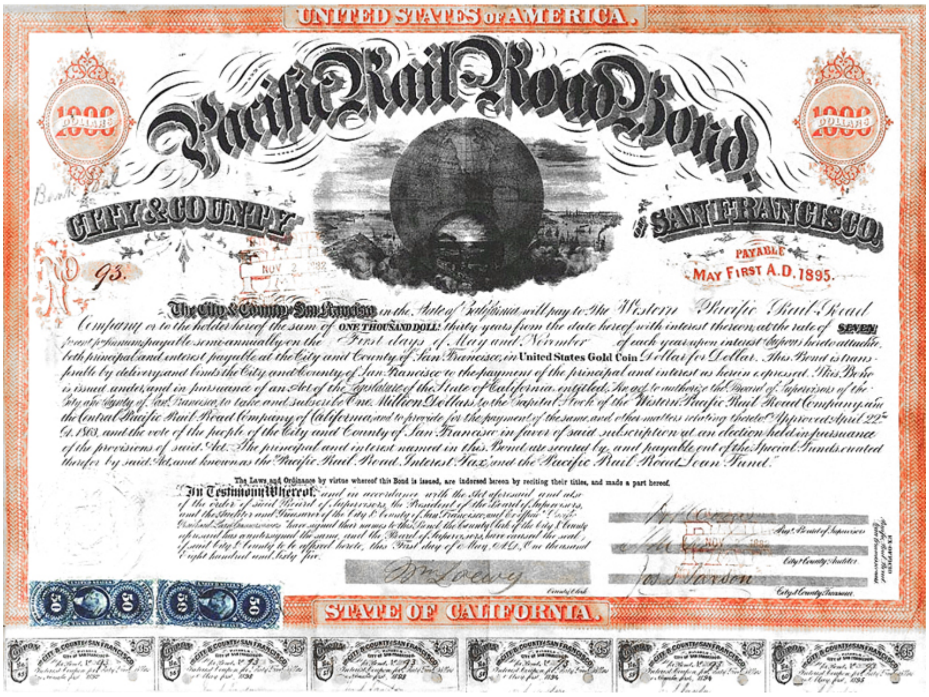
- **a set of coupons** of fairly definable dates and payments, which from a programmatic perspective seemed quite tractable, and
- **a set of terms and conditions** which I felt also could be modelled, indeed which intuitively I felt was easy to model.

It may strike the reader at this point that choosing bonds was naïve and misinformed. In retrospect, it was naïve, as the ease of entry was only superficial. In practice, corporate bonds were a big boys game—only the largest corporations issued bonds, and nations and supra-nations were the favourites. Banks and brokers were the intermediaries, and unlike today, the holders of last resort. All the players were quasi- or directly-regulated in some fashion, so this was a market easily defended.

I was to find that not only would a fintech upstart not be able to enter this market, I'd have to create an entirely new market – e.g., small bonds to small companies. Inept indeed for a self-claimed student of strategy, but this mistake was critical to that which followed.

Deep dive

[5] Gary Howland, "The Development of an Open and Flexible Payment System", 1996 http://systemics.com/docs/sox/overview.html



UNITED STATES OF AMERICA.

Pacific Rail Road Bond.

1000 DOLLARS

CITY & COUNTY OF SAN FRANCISCO.

No 93.

PAYABLE MAY FIRST A.D. 1895.

The City & County of San Francisco in the State of California will pay to the Western Pacific Rail Road Company or to the holder hereof the sum of ONE THOUSAND DOLL.S thirty years from the date hereof with interest thereon, at the rate of SEVEN per cent per annum payable semi-annually on the First days of May and November of each year upon interest Coupons hereto attached, both principal and interest payable at the City and County of San Francisco, in United States Gold Coin Dollar for Dollar. This Bond is transferable by delivery and binds the City and County of San Francisco to the payment of the principal and interest as herein expressed. This Bond is issued under, and in pursuance of an Act of the Legislature of the State of California, entitled "An act to authorize the Board of Supervisors of the City and County of San Francisco to take and subscribe One Million Dollars to the Capital Stock of the Western Pacific Rail Road Company and the Central Pacific Rail Road Company of California, and to provide for the payment of the same, and other matters relating thereto" Approved April 22nd A.D. 1863, and the vote of the people of the City and County of San Francisco in favor of said subscription at an election held in pursuance of the provisions of said Act. The principal and interest named in this Bond are secured by, and payable out of the Special Funds created therefor by said Act, and known as the "Pacific Rail Road Interest Tax," and the "Pacific Rail Road Loan Fund."

The Laws and Ordinance by virtue whereof this Bond is issued, are indorsed hereon by reciting their titles, and made a part hereof.

In Testimony Whereof, and in accordance with the Act aforesaid and also of the order of said Board of Supervisors, the President of the Board of Supervisors, and the Auditor and Treasurer of the City & County of San Francisco, and Ex officio Pacific Rail Road Commissioners have signed their names to this Bond, the County Clerk of the City & County aforesaid has countersigned the same, and the Board of Supervisors have caused the seal of said City & County to be affixed hereto, this First day of May, A.D. One thousand Eight hundred and sixty five.

[illegible] County Clerk

[illegible] Pres.t Board of Supervisors

[illegible] City & County Auditor

[illegible] City & County Treasurer

STATE OF CALIFORNIA.

Figure 1: Pacific Rail-Road Bond, issued 1985. Source: wikipedia.org

I then turned to investigate what complexity existed in a bond, because I reasoned that we had to capture all that complexity into our system. In those days, bonds on paper still existed, and it was possible to identify the basics more easily by looking at the old paper-issued debt instruments. A typical bond is a document with three major components:

- **A set of parameters:** an issuer, a face value, a date of payment, an amount for interest payments, a schedule. These were regular and easy to model in computable-name, value pairs.
- **"Coupons":** dated payments of interest. Finance classes had taught me that each coupon was just another zero coupon bond and therefore, if I could figure out the general case of how to issue a zero coupon bond problem, then I'll have solved the problem for coupons as well.
- **The fine print**: the terms and conditions that varied the basic or standardised form.

I declared the first and second parts easy, which left the fine print. As a computer scientist, I was convinced that the field of law was overdue for disruption, and that clearly we could model the content into some form of database layout or domain-specific language. Especially when the terms were clearly laid out, as on many older bonds. Most modern bonds, however, had been computerised, and had adopted much more complex prose for their terms. The model was the same, but it seemed that, as we left behind the discipline imposed by needing to fit it all on a single piece of paper, the quantity of clauses exploded.

At some point, I realised that a bond was nothing more and nothing less than a contract, which was serendipitous because—although all financial instruments are contracts at heart—only the bond so clearly presents itself in this way. Contracts were a new mystery, but they too were modelled by lawyers as a set of "elements":

- A party and a counterparty,
- An offer and an acceptance,
- Consideration of goods or services in one direction balanced by considerations of cash in the other,
- Terms and conditions.

I felt that the complexity of finance was unravelling: there was a contract agreed at the beginning, and there was performance of that agreement to follow. This separation of the lifecycle of the bond into two separate spaces led to a glimmer of hope—if we could render that division into code, dividing agreement and performance would make the job of each of the halves so much easier. Understanding the bond contract then appeared as a (complicated) task of reducing legal English into a database of fields. I charged into terms and conditions with gusto. I would reduce the whole lot into tag-values or SQL[6] tables or expressions or language or something. Then, the rest of the accounting system that Gary Howland was building could handle the performance from all this information. This I could do because I was a Computer Scientist!

Resistance level

Yet, the bond resisted. The clauses were not that clear, nor standardised, nor even settled. I discovered that each and every issuance of debt copied from its forebears, but each new bond was an opportunity to tweak, to fix, to turn the conditions for some kind of incremental benefit. Bond legal prose tended to grow over time, in ways that we could not predict; no two bonds were directly equivalent. This was an important challenge—why would anyone trust their valuations to a new cryptographic system that failed to capture all that was in a bond?

I also discovered a pattern of fighting the last war: every bond debacle would result in new clauses. The cynical amongst us would say the

[6] SQL is a domain-specific language used in programming, typically for relational databases.

lawyers were turning opportunity into fees, but it is human nature to tinker, and indeed there are always improvements to make. Worse than continuous evolution, I got the sinking feeling that the use of language was combative—it could be used to hide asymmetries which favoured the issuer over the holder. My unfortunate conclusion, then, was that it was impossible to convert all of the English language as used in bonds into a data language or schema so long as it was under the Byzantine influence of legal wordsmiths.

2. The War of the Wordsmiths

That left some options. Could I get rid of the wordsmiths? No—not in my lifetime, nor the lifetime of our startup. Would a suitably good algebraic formulation render the English prose redundant? No—for better or worse, some of the concepts could only be explained in human language for reasons of vagueness, uncertainty, unpredictability or even low probability. Why spend pages over an event that only happens once in a blue moon? It is efficient for the clauses to be incomplete wherever the future is both unknown and a contingency is unlikely. Yet, for us programmers, incompleteness in English is a showstopper. Could I convert some of the contract? The most part of it? The important part? Even the answer to this question appeared to be "No"!

I had a sinking feeling that rendering the contract into computer language would be a bad thing. If the English language were being used as a battleground between the parties, then the use of the languages and data structures of computing would simply add more power to their weapons. Unless the conversion was a perfect one between English and a computational language, I would simply be adding more room for more trouble. These were consequences I did not want to foresee; I did not want to be the expert witness called to court to explain why the computer language said the opposite of the legal prose.

And this applied to even the simple elements of the contract—the face, and the issuer for example. If there was duplication of any form, there was even more room for trouble.

At that point, perhaps, a principle emerged: Simplify! Marketing classes taught me that consumers paid double for a simple product. Security thinking told me that more complex models result in more insecurity. At that time, I was becoming suspicious that systems of Internet security were failing because they thrust too complex a model onto the poor user. Simplicity not complexity was the key to security, something I later captured in the aphorism—*there is only one mode, and it is secure*—which I later found out to be embedded in the 6th principle of Auguste Kerckhoffs exposition of military cryptography:

> *Finally, it is necessary, given the circumstances that command its application, that the system be easy to use, requiring neither mental strain nor the knowledge of a long series of rules to observe.*[7]

2.1. The contract is the issue

At some point, the inspiration hit me to flip my logic upside down. Keep the document as is—in legal English in all its juridical glory—and let the computer do the work of extracting the information it needs *after the fact*. Instead of converting up front for ease of programming, convert lazily for ease of contract writing. We fight for the users, not the developers.

Suddenly, it all made sense. The document had to be readable primarily by humans, and only secondarily by the computer. I had been wrong to want to place the computer first, in part because I'd been seduced by the financiers' drive to value the financial instrument, and by the developers' drive to "automate all the things." But my goal had never been that, it had always been to issue a financial instrument. Valuation was the task of users; let them do the hard lifting, let the marketplace find a way to reduce a contract to a database.

Embedding 10 or so fields into the text in computer-readable fashion was an easy problem, whereas getting humans to work with computer layout was intractable. The contract for digital issuance had to be in plain human language, and everything else followed.

A practical computer-contract places human first

My goal then became to tweak the document to capture the variations, but eliminate the complexity, so that Gary Howland's core payments or accounting system could just deal in quantities and identifiers. This consisted of three steps: (i) rendering the contract itself—warts and all—into a single, readable, and signable digital document; (ii) making the performance aspects readable by the client software in some sense; (iii) identifying each instrument, so that we could offer a system with many thousands of bonds in play at any one time. I will take these steps in turn.

[7] Auguste Kerckhoffs, "La cryptographie militaire ('Military Cryptography')," Journal des sciences militaires, vol. IX, pp. 5-38, Jan. 1883, pp. 161-191, Feb. 1883.

A signed digital—readable—document

In the early 1990s, a software tool called PGP — standing for *Pretty Good Privacy* — provided secure mail encryption and identity, and it showed us how to sign a document in plaintext. Tantalisingly, in those days we saw our architecture as extending the basic email encryption system of PGP into finance. Indeed, first we wrote Cryptix, the original Java and Perl packages for cryptography.[8] Then we implemented PGP over the top of Cryptix, and only then did we construct Ricardo as a payment system on that stack. The first system's identities were actual PGP keys, and instructions were PGP-signed records, so you could in theory use your PGP email identity to issue and transfer bonds and money within Ricardo. It was natural and easy to express the contract as a PGP-cleartext signed document, a feature it already had.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

05.02.11
'Alice' is the owner of the GPG key with fingerprint:
4F16 E4D6 BB9B D4A0 39F8  9644 DF23 CB88 2400 ACE3
'Bob' is the owner of the GPG key with fingerprint:
05CA A3B0 9322 1874 9D1A  2357 9C07 2DDC 4394 91B7

This contract is for the exchange of 20 Bitcoins at a
rate of USD $3.25 per bitcoin, for a total of $65 USD.

Bob agrees to send $65 USD, plus any fees charged by
Paypal, via a Paypal payment with transaction type 'Payment
Owed' (to reduce chargeback risks) to the paypal account
'alice@lol.com' within 24 hours of both parties
signing this contract. Alice agrees to send 20 bitcoins
to IDj1SocbbH9Lbb9aTdqSHB9AAjhdxNNZha within 4 hours of
receiving this Paypal payment.
-----BEGIN PGP SIGNATURE-----
Version: GnnuPG v1.4.11 (GNU/Linux)

iJwEAQECAAYFAk2/PKAACgkQ3yPLiCQArOOc/AP9GL0EgVQMTHZqOX5ynNVGBFb2
6eB7QzRdNQH8Zcj6R0y7fzbpYPbgwX+G3EYtsDjS4G3M8Ld1FFCcJ/JLJGle191e
KLpXp/BWMRayn3KcFYoGogmONtxk1wOVoXF+wiK9jZYFIdjI87qh8iUOCboFVqQk
T3OG7odEKJOjNwYP+j0=
=2mDw
-----END PGP SIGNATURE-----
```

Figure 2: a stylised contract to swap signed with PGP.
(Prose in grey Italics, PGP signature artifacts in normal)

A holder of an instrument expects the identity of an issuer to sign the contract, so we also put the issuer's certificates inside the contracts. As trust in finance is based on personal knowledge of people and own due diligence,

[8] Our Cryptix was the most popular crypto library for Java until the arrival of the abomination known as JCE.

we were able to use PGP identity all the way, and avoid the requirement common in other systems for a gatekeeper—for example a regulator or industry association that permitted newcomers into their systems via a process that could be described as due diligence or discrimination, variously.

Markup

The software would also need to extract the useful information from the readable document. How much information did users need? If investors were analysing a bond for pricing then we would need a lot of information; but if bankers were building an accounting engine, which is the foundation of a digital cash and bond trading system, abstractly, we would only need a small set - about 10 simple fields. These would include the name of the issuer, the type (e.g., debt) and name of issue, the face amount and currency and coupons, and so forth. Simple!

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

;
;  Prepaid Services Dollar, Issue A.
;
;  Being, a Contract to settle USD-denominated services.
;
;  Between, Systemics Inc. and Users.
;

[definitions]

definitions_dollars = *
{
  Prepaid Services Dollar ("PSD") means the electronic
  currency, denominated in United States of America dollars
  ("USD"), as facilitated by this Ricardian contract.  Other
  dollars, which may be used as exchange for PSD, are referred
  to as Account Dollars.
}

definitions_units = *
{
  The unit of the PSD is the iota, which is defined as having
  the value of PSD 0.0001.
}
```

Figure 3: snippet of Ricardian 'stablecoin' dollar contract showing explanatory comment at top, heading and prose clauses [9]

[9] Systemics Inc's Pre-Paid Services Dollar Ricardian Contract, 2003, http://webfunds.org/ricardo/contracts/systemics/Systemics_PSD_a.html or also see https://iang.org/rants/systemics_psd.html

I asked Gary Howland to knock up a simple markup language. I thought he'd give me HTML-for-finance, but instead he gave me the venerable old INI-format from Microsoft. I asked for a 2-dimensional structure, he added headings.

But he was right and I was wrong; the simplicity of the line-based, *tag=value* pairs did us noble service and every attempt to "update" the format to advanced miracles such as XML gave poorer results [10]. With our format of what we would now call a simple markup language, the contract became parseable by the client software for the cost of about 1000 lines of code—a trivial amount. Instead of trying to do everything, for all circumstances and all users, we just did what we needed, and found that we only really needed about 10 fields to do almost everything.

```
[issue]
;
;  This section identifies general aspects of this contract.
;
issue_type = currency
issue_name = Systemics Pre-paid Services Dollar

[currency]
currency_symbol = $
currency_tla = PSD

[unit]
;
;  The Unit of Account is the PSD. This currency is denominated
;  in PSD, with an underlying unit of contract of iota, which
;  is equal to PSD 0.0001.
;
unit_power = 4
unit_mediate_power = 2
unit_major = $
unit_mediate = c
unit_minor = p
unit_major_unit = PSD
unit_mediate_unit = cent
unit_minor_unit = iota
unit_major_units = PSD
unit_mediate_units = cents
unit_minor_units = iotas
```

Figure 4: snippet of Ricardian 'stablecoin' dollar contract showing tag-value pairs and slightly smart decimalisation[11]

[10] Erwin van der Koogh, "Ricardian Contracts in XML," 2001 Edinburgh Financial Cryptography Engineering 01 Conference, 22rd and 23th June 2001

[11] Systemics Inc, *op cit*

The identifier

I had, by then, done a lot of reading on the forms and theory of bonds. We were in an era of the digitisation of bonds, sometimes called dematerialisation, which was a fancy word for getting rid of the paper. One thread I had noticed in the evolution of the times was that, as soon as any form of automation was required, the IT people insisted on a unique identifier.

And woe betide the bond that presented itself with the same identifier as another bond! Or a format that wasn't acceptable! Or from an unauthorised source! What had previously been a liberty of capital was rapidly becoming an activity controlled by bureaucrats in charge of numbers.

These were early days in globalised computing, and we really only had one way to create an identifier: ask a centralised group to allocate numbers. In the context of bonds, it had to be a national body to allocate the numbers.

I rebelled. Going back to my original choice of bonds as a perceived "easy entry market", I didn't want a bond system which was beholden to a remote and uncaring agency that sought to extract easy rents. Easy entry was broken if some national organisation were in charge of the numbers. And what about international bonds? The biggest bond issuer of all was the World Bank, and if it were to be an Internet bond system, it had to be able to perform for all issuers. I wanted an exchange like the old Amsterdam exchange, where anyone could walk in and do trade. No permissions, no controls, *caveat emptor*. At the very least, I did not want some self-appointed committee of number-allocators to provide those controls through accretion of power.

Clearly, any system of many components needs naming. These were the exciting times of the Web, and things were being named by URLs, IP numbers, and domain names. Each of these systems exposed features and pitfalls, many of which were mirrored in finance—numbers and ticker symbols allocated by institutions were like domain names of companies, intellectual property to be battled over because of artificial scarcity.

I wanted none of that. I could already see that the naming systems for domains, IP numbers and URLs were creating headaches for users. Control, scamming, theft were erupting around these areas. Costs! I feared the death of a thousand cuts would kill the then open business of the Internet. Even Java had chosen to base its international class system on domain names; as we were engaged in the Crypto Wars, we did not follow Sun's lead and named all our classes rooted in Cryptix not org.Cryptix. I wanted a system which did not demand permission at any point, and left open no side-channels for attackers to exploit.

Although I didn't say it in these terms, I wanted a system of contracts that could not be stopped by anyone — a principle that was later to drive Bitcoin. And, there was a simpler solution.

The hash as identifier

PGP again provided the answer—as it had a convenient feature of providing a cryptographic message digest or *hash* of any document. Out of my paranoia emerged the idea that a hash as an identifier for a contract cut through a lot of costly nonsense. In those youthful days of open cryptography, most people talked about the cryptographic message digest as a compression function for use in RSA digital signing, but we knew that hashing could be used to create a digital identifier for any document, not just a signature. It's just perhaps that nobody really did that at the time.

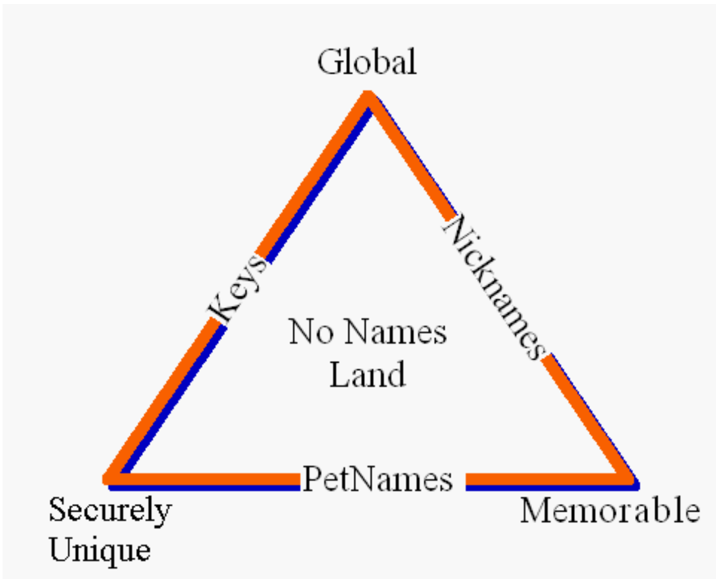


Figure 5: Zooko's triangle – the Ricardian hash provides for 2 of the trilemma of names, being global, and securely unique, but hashes lack memorability

Beyond all, the hash was self-generated, self-proving, and guaranteed to be globally unique and secure—the memorable name would be easily included in the contract to which the hash pointed [12]. That final part was perhaps the most innovative of the design, perhaps because we had tagged legal contracts with cryptography. Or so we felt at the time.

The form of the contract

And so it proved. The Ricardian Contract emerged as an INI-formatted prose contract with about 10 important embedded tag-value pairs, signed by the PGP key of the issuer, and hashed with SHA1 as an identifier.

With this design pattern, we had achieved the perfect separation between the world of law and the world of accounting - contract on the left, transactions on the right, and the hash both dividing and joining them. What's more, as a legal document, it could describe anything a contract could describe, and as a cryptographic contract, it could be issued, traded, and valued without limits imposed by the technology. It quickly became clear that the Ricardian Contract was the right building block not only for bonds, but practically all of finance; bonds had just been the lucky one to most clearly surface the contract, and thus put us on the path to dividing finance into a legal contract on one side, and an accounting system of numbers on the other.

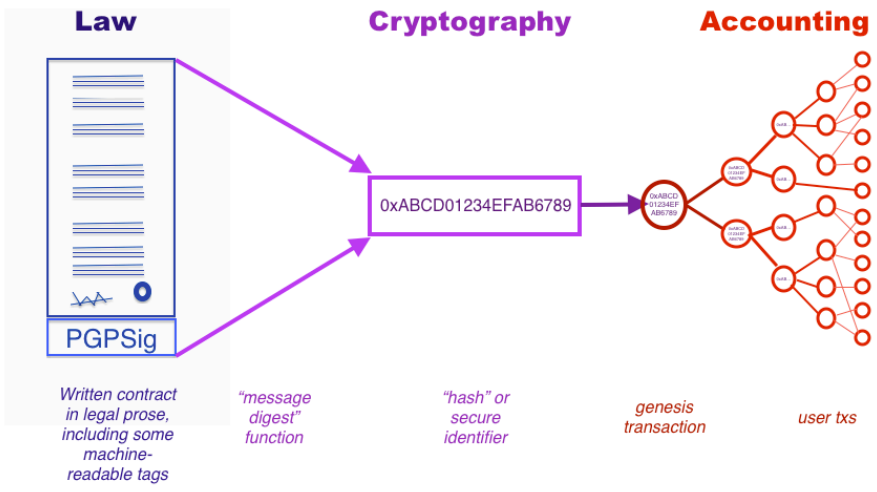

Law
Cryptography
Accounting
PGPSig
0xABCD01234EFAB6789
0xABCD 01234EF AB6789
Written contract in legal prose, including some machine-readable tags
"message digest" function
"hash" or secure identifier
genesis transaction
user txs

Figure 6: the Bow-Tie model of the Ricardian Contract

[12] Bryce "zooko" Wilcox-O'Hearn, "Names: Decentralized, Secure, Human-Meaningful - Choose Two", 2001; and

Marc Steigler, "An Introduction to Petnames," 2005 http://www.skyhunter.-com/marcs/petnames/IntroPetNames.html

2.2. Random Experiences on the Ricardian Journey

Commercial success did not happen to us. We got close, but it is a curse to be ahead of one's time, as it is simply too hard to explain to investors and buyers. Further, the Ricardian Contract had the effect of forcing brutal honesty on issuers, which wasn't a selling point in an industry where every second startup in the 1990s was a Ponzi in disguise (the ICO experience of the late 2010s was worse). However, we built up a lot of experience in the issuance of digital contracts, and that's worth recording. Some quick snippets follow.

- **Form.** The layout should be as readable as possible. It also needs to support digital signatures and hashing, and in practice this means canonicalisation. These two requirements are contradictory, and lead to a third: simplicity! In practice, this makes XML a bad choice, but it is also the most popular choice.
- **The One.** In time, we established a wider view of goals and characteristics of the Ricardian Contract.[13] As part of that, we also established *The Rule of One Contract,* that there be only one form of each contract. Every time we tried to improve the results—databases, intermediaries, etc—confusion was the result. You can pick any variation you like, but my advice is to stick to the One!
- **Acceptability.** Our early fear that the legal fraternity would reject our novel contract was unfounded. Courts love to see exotic contracts because it brightens up their dull days, and it allows character to flow through and reveal what is really going on.
- **Arbitration.** One Ricardian Contract landed me into trouble because it had a clause referring disputes to Arbitration, which referral I tried to fight. And lost—but the experience of the following arbitration was very informing, and it taught me an important lesson. Certainty of dispute resolution at the end of the process was just as systemically important as certainty of the contract at the beginning of the process. Dispute resolution has wider effect than just contracts. In CAcert, a world wide open community certification authority (signer of web certificates), we built an arbitration component for all purposes, and it lifted the entire organisation to a higher level of reliability.[14] From that experience, I theorise that arbitration can be seen as the apex of an inverted pyramid—over which we layer deep support, user support and finally

[13] Webfunds Project, "Ricardian contracts," notes for developers http://webfunds.org/guide/ricardian.html

[14] Ian Grigg, "An Open Audit of an Open Certification Authority," 22nd Large Installation Systems Administration Conference (LISA 2008) 13th November 2008 https://iang.org/papers/open_audit_lisa.html

business; the trade supported is only as strong as the apex: the resolution of disputes generated by the trade.[15] Which experience offers itself as a clear benefit for blockchains built for trade and trust, a topic I return to below.
- **Innovation?** Gary Howland and I were pretty convinced that everything we had done was just good software engineering, including all I've described in this article. We expected others to follow suit as soon as they walked the same path. So much so, we didn't even name it; which is the humbling story behind the name: it was the contract in Ricardo, that's all. It took me years to realise how innovative the Riccy was, as the finance people tended to call it. We never patented it, we never copyrighted it, nobody said "you should write a paper about that…" Hence the original paper only followed years after, as an afterthought derived from rough notes after an attempt to explain it to other engineers.
- **Regrets?** The Ricardian Contract solved its design problems in the day and did so well—there have been no regrets over the original design. Yet, the world has moved on and the environment of today is very different to that of the mid-1990s. Several developments have pushed the original design into new territory.

Contracting, not contracts

Chris Odom was the first to fully adopt and extend the Ricardian Contract in his system OpenTransactions.[16] He extended it in three ways:[17]

- To use XML;
- To allow one Riccy to include another, a construct he called "Russian dolls";
- To use the form for many purposes: messages, datafiles, ledgers, payment plans, markets, and trades.

My view and purpose had been the *single issuer, multiple holder* context of an issued financial product; Odom's insight broke that assumption completely open. The Ricardian Contract was now capable of handling any variation of party arrangements—a Riccy could be a component or packet

[15] Ian Grigg, "The Inverted Pyramid of Identity," FC 2009, http://financial-cryptography.com/mt/archives/001165.html

[16] Chris Odom, "Open-Transactions: Secure Contracts between Untrusted Parties", http://www.opentransactions.org/open-transactions.pdf Downloaded 2016.06.06.

[17] Chris Odom, "Sample Currency Contract", http://opentransactions.org/wiki/index.php/Sample_Currency_Contract Downloaded 2016.06.06

leading up to a wider agreement. Odom's insights were later adopted by OpenBazaar in their system of user-to-user purchasing. Each phase was a signed Ricardian Contract in JSON that included the previous phase:

- Vendors invite shoppers to treat;
- Shoppers offer to buy over the invitation;
- Vendors accept over the offer; and
- Delivery results in payment over the acceptance.

2.3. Enter "Smart Contracts"

The novel ideas of Nick Szabo percolated through the cypherpunk community of the 1990s, but seemed stuck in the domain of abstraction.[18] At that time, every problem that smart contracts seemed to solve could either be better solved by coding in features into the clients and servers, or had low demand. As a trivial example, the Ricardian Contract had always sported a very limited form of "smarts" in the form of decimalisation, code for which was shared between the contract layout and the client-side display code.

The innovation of Bitcoin changed all that. Instead of providing a raw hard-coded payment, Bitcoin operates by verifying small programs written in a low level language akin to CPU instructions, derived from an old virtual-machine language called Forth. Loosely, one validation rule states that numbers must already exist, another states an exception: the first program in a block can creates some numbers. Different programs can be written to envisage higher forms of transaction. Being a shared computation system rather than, narrowly or strictly, a payment system, carries some legal and regulatory implications: the data as a result of the shared computation bears little evidence of e.g., payments, and it can only be determined to be a payment system in light of the use made by its users.

No matter that all the popular forms—multisig, crowd funding, time-locks—were still more efficient and tractable in hard-coded form, the smart contract excited an entirely new generation of financial cryptographers. It is fair to say that Bitcoin offered the promise of smart contracts, but delivery was uncertain. Thus sparked an open competition to deliver the first wide-spread scalable smart contract platform, by Bitcoin itself and others such as Ethereum and R3's Corda.[19]

[18] Nick Szabo, "Smart Contracts", 1994 http://szabo.best.vwh.net/smart.contracts.html

[19] Richard Gendal Brown, et al "Corda: An Introduction," R3 whitepaper 2016

It is also fair to say that the Ricardian Contract solved a very different problem to the Szaboian smart contract: the Riccy captured the *legal content* at the offering, made it readable and displayable for the holder to accept, and provided a great identifier for all uses. The "contracts" of the blockchain world, on the other hand, were concentrating on performance—after the contract was agreed. These are not, then, competing ideas at all, they are ideas aimed at different phases of the contract life-cycle.[20] The real question was how to combine all of these ideas together.

The Bitcoin Unit

One of the oddities of Bitcoin is why it did not use something like the Ricardian Contract. The answer to this, I believe, lies in the nature of contracts. In our world of the 1990s, we believed we could sell software to issue bonds and cash and all sorts of financial instruments, and our design needs were to identify the nature of our instruments as much as possible.

This was not how financial cryptography turned out. In practice, no financial instrument other than money took hold. In contrast to our aspirations of strong governance, we saw a steady series of quasi-money issuers who concentrated on fairly simplistic claims that did not come anywhere near to the strength of the Ricardian Contract: Paypal, Webmoney, e-gold, goldmoney, EFTs and Liberty Reserve are some of the better known names.

I include names of various shades for a reason. In contrast to our designs of strength through information, many systems preferred strength through obscurity, and many of these ran into legal (including criminal) trouble. It was this trouble—the persistent failure and disruption in the presence of attackers of all forms—that inspired the design of Bitcoin.

Satoshi Nakamoto designed a system that could not be shut down. By its inherent logic, such a system could not have an issuer, nor an underlying value in payments as mentioned above, nor delivery of coupons nor dividends nor options nor any other promises. In short, all of the elements of the contract threatened the mission of Bitcoin, and consequently the Ricardian Contract or any similar contractual form had no place in Bitcoin. But, as an unforeseen consequence, this also suggested that there could be only one currency per chain, because as soon as you had two, you had to describe the difference.

The sum of all chains

[20] Ian Grigg, "On the intersection of Ricardian and Smart Contracts," working paper 2015 https://iang.org/papers/intersection_ricardian_smart.html

Bitcoin may have only one unit, but it has spawned a frenzy of "alt-Coins"—copies with minor variations in contractual and parameter terms. Of course, these lacked Ricardian Contracts, because there was none such in the original Bitcoin, but they still required description in some fashion.

Vaguely, this difference was described by websites and chat rooms and word of mouth, but more importantly, the software also had the problem that it had to change to accommodate the mostly trivial changes between the original Bitcoin and a new altCoin design—including the location of new chains. As crazy as it seems, all of the details for chains, including the genesis block, are hardcoded into the base software distribution, and the process of issuing a new altCoin is one of hacking the source code and making every new customer download a new program with the new parameters hardcoded in.

This immediately struck me as a Ricardian problem, so I designed a variant that could specify the details of the chain. This tinkering would allow one client software to manage multiple chains from the same code base, just by reading the chain "contract". It made sense to me—if you, as a company, were selling the services of coffee-chain or bond-chain or realty-chain, you would want to do this within a contractual framework, notwithstanding the original Bitcoin innovation.

If a legal context made sense, so did parameters within the chain's genesis transaction; things like the location of seed nodes, alert keys, the time of a block, and the infamous block size limit all need to be coded into a descriptive device, and in some cases they might need to be adjusted over time. And, if Bitcoin's mining schedule isn't screaming *smart contract* at you, then you've attended too many loud Satoshi rock concerts.

Unchaining the chain

So emerged an augmented Ricardian Contract or *Ricardian triple* to describe a chain with {prose, code, params} and, of course, using the now-familiar hash as identifier[21]. What was interesting was that this extended design pattern could also describe a wide range of things: smart contracts, individuals, corporations, and even devices or nodes could all be described by a tuple of {prose, code, params}. These devices are more like network-empowered objects in object-oriented thinking, or capabilities. Indeed, there may have

[21] Ian Grigg, "The Sum of all Chains - let's converge," 2015, http://financial-cryptography.com/mt/archives/001556.html

Journal of Law, Information and Science *Vol 2X(X) 201X-201X*

been two forerunners to this augmented form, being the E language[22] and the Askemos system.[23]

The governed blockchain

Drawing from the above experiences, the Ricardian Contract was introduced at the genesis point in the launch of the EOS blockchain.[24] Block producers agreed to a (Ricardian) Community Agreement, and then required agreement by all users in transactions on the chain. The written agreement included clauses on changing the document, and resolution of disputes in a community forum.

This was referred to as the *governed blockchain*. [25] This design was not without gaps, and a key difficulty was showing that users had entered into the agreement and thus the jurisdiction.[26] Sadly, adverse elements within the community managed to neuter most of the governance just as it was starting to deliver results: rulings on million dollar cases and community-driven funding for essential works were blocked, burnt, disassembled. Other blockchains have now built successful arrangements of governance, and kept them.

Conclusion

This brings us to the current day. In conclusion, I will mention five points for future research:

The tuple. A team at Barclays Bank is building out advanced Ricardian Contracts as laid out above with a tuple of {prose, code, params} towards a long-term goal of replacing the constellation of ISDA swaps contracts.[27] It is

[22] Mark S. Miller, Chip Morningstar, Bill Franz, “Capability-based Financial Instruments - an Ode to the Granovetter Operator.” Financial Cryptography 2000, Anguilla

[23] Jörg F. Wittenberger, “Askemos - a distributed settlement.“ 2002 http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.5050

[24] See https://eos.io.

[25] Ian Grigg, “The Governed Blockchain,” block.one white paper 2018 https://iang.org/papers/the_governed_blockchain.html

[26] Adam Sanitt and Ian Grigg, "Legal analysis of the governed blockchain,” NortonRoseFulbright and block.one working paper 2018

[27]Clack, et al “Smart Contract Templates: foundations, design landscape and research directions,” Barclays working paper 2016 https://arxiv.org/abs/1608.00771

clearly an interesting question to see how far we can extend the idea of the tuples—can we identify everything this way?

Prose versus Code. The horizontal integration between code and prose is being thought about in several ways:

- to pair tight tuples of code fragment with prose fragment together (CommonAccord);
- to derive the code from the prose or vice versa;
- to verify conformance of code to prose or vice versa; and
- to insert the code into prose or perhaps vice versa.

The last is my favourite, because it conforms to the rule of one contract.

It should perhaps be added that I believe that stronger integration of code and prose should be viewed with caution. On the one hand, some form of logic is required to for example handle coupons, and simple functions or procedures in a high level language such as Java could be inserted into the prose. Our work in simple issuances did not reach that need.

In contrast, issues found in prose such as fuzziness, ambiguity and incompleteness are rarely of interest to the user and therefore of little interest to the program - we don't need to code up what the user does not use. These factors tend to become of key importance when a dispute happens, but disputes are rare, and are resolved by humans who can read. To this extent, I believe interpreting pure prose as code or vice versa may be a red herring.

Enforceability. There is a question of whether the Ricardian Contract can be enforced, and whether a court will respect the tradition. I think this is so, but I am not a court, and there is an open question of examining precedent here. As much of blockchain and financial cryptography practice is international and cross-jurisdictional by nature, I suspect the winning solution here is contracts that refer disputes not to courts but to international forums of arbitration that accept and align with the community practices assumed in the contract.

Contract Browser. Finally, a challenge which we anticipated and tried to surmount but never achieved: there is a crying need for a contract preparation tool. The workflow and negotiation of contract formation is opaque to the technologist, and verification of format, key management and signing are technical tasks beyond the non-technical lawyer. A careful melding of the right skills is needed.

Identity. My own work is in extending the Ricardian formula to document interactions between peers.[28] I see these events between people as the next generation of identity systems, as they more comfortably capture the persons as they are, as they act and as they wish to control their data.

[28] These events are called variously attributes, verified claims or reliable statements (my favourite). See *inter alia*, Rebooting Web of Trust, https://www.weboftrust.in-fo/