Cracking the Agrippa Code: Cryptography for the Digital Humanities

Scholarly and Research Communication
VOLUME 4 / ISSUE 3 / 2013

Quinn DuPont
University of Toronto

Abstract
In *The Laws of Cool,* Liu (2004) argues that the art book *Agrippa (A Book of the Dead)* (Gibson, 1992) is an exhibit of destructive creativity. According to Liu, the book's great *auto-da-fé* occurs when the software program, which is included with the book, displays an electronic poem, and then self-encrypts, a mechanism that destroys or "permanently disappears" (p. 340) the poem. This article argues that Liu's understanding of encryption is incorrect. Encryption is not destruction because enciphered text is necessarily subject to cryptanalysis ("cracking"). Relatedly, this article demonstrates that Kirschenbaum's thesis of "no round trip" is mistaken (Kirschenbaum, Reside, & Liu, 2008). *Agrippa* was fully cracked and reverse-engineered in the course of an online, global cryptanalysis challenge. This article describes the forensic details of *Agrippa* and its cryptographic routines.

Quinn DuPont is a PhD candidate in the Faculty of Information at the University of Toronto, Canada. Email: quinn.dupont@utoronto.ca .

Keywords
Cryptography; Code; Art; New media art; Crowdsourcing

In 1992, cyberpunk author William Gibson was commissioned to write a short poem to be included in a noir art book published by Kevin Begos, Jr. and designed by Dennis Ashbaugh. The result was *Agrippa (A Book of the Dead)*. This lavishly decorated book contains copperplate aquatint etching of simulated DNA gel electrophoresis, long DNA sequences from the bicoid morphogen gene of the fruitfly, and a number of faded

CCSP Press
Scholarly and Research Communication
Volume 4, Issue 3, Article ID 0301126, 8 pages
Journal URL: www.src-online.ca
Received May 31, 2013, Accepted March 8, 2013, Published June 3, 2013

DuPont, Quinn. (2013). Cracking the Agrippa Code: Cryptography for the Digital Humanities. *Scholarly and Research Communication, 4*(3): 0301126, 8 pp.

vintage advertisements (the book was published in two versions, the so-called "small" version being less elaborate) (Liu, Hehmeyer, Hodge, Knight, Roh, Swanstrom, & Kirschenbaum, 2008). These material furnishings portray the ubiquity of codes, the way they come into being, and how they are forgotten. Additionally, a 3.5" diskette was embedded in the back of the book. This diskette contained a Mac System 7 program that, when run, scrolled Gibson's poem on screen. The poem, in keeping with the motif of the book, tells about memory, loss, nature, and mechanism, all framed by a Kodak photo album.

When released in 1992, the poem attracted considerable attention, but due to the extremely limited print run, very few people have seen the book or the poem first-hand. In an interesting twist of fate, a transcript and then a video recording of the poem surfaced online (Kirschenbaum, 2008). These early leaks (surreptitiously recorded) came from a public showing of the software, known as the "The Transmission," held in The Kitchen, an art space in New York City. For over a decade this was the only source of information about *Agrippa*. In 2005, Alan Liu and a team of graduate students created the scholarly site, *The Agrippa Files*, and working in collaboration with Matthew Kirschenbaum at the Maryland Institute for Technology in the Humanities and the Digital Forensics Lab, recovered a bit-for-bit copy of the application, along with numerous archival documents detailing the production of *Agrippa* (Liu et al., 2008).

This article lays the groundwork for correcting a common misconception of cryptography. Liu calls *Agrippa* an "exhibit" of his thesis of "destructive creativity," in large part because he believes the cryptographic mechanism in *Agrippa* erases the poem once it is displayed. I will show that *Agrippa* is not destroyed when run, inductively pointing to the conclusion that cryptography necessarily excludes destruction (a conclusion suggested here, but not fully developed). Additionally, I will show how *Agrippa* does not self-encrypt, although a particular kind of self-destruction does occur. This strange self-destruction leaves the ciphertext untouched, so that cryptanalysis of the poem is possible even after the disk has been run. Contra Kirschenbaum, I argue that *Agrippa* does indeed "round-trip" (Kirschenbaum, Reside, & Liu, 2008).

By revealing the forensic details of *Agrippa*, I will show that cryptography is a rich field of study for digital humanities. I believe that cryptography has been hitherto ignored by digital humanists because it has been thoroughly black-boxed (Latour, 1988), typically construed in technical terms as a form of "secrecy" (Shannon, 1949). This view is historically anachronistic – cryptography has a long and diverse history – and it is better conceptualized as a special kind of writing system (an important issue, but a matter to be taken up elsewhere). *Agrippa* is a particularly rich example and has received considerable attention already (Liu et al., 2008), but it is one object in the species of cryptographic works, and such works have traditionally been ignored by the humanities. I show how collaborations (in this case, crowdsourced collaboration) can interrogate cryptographic objects and yield a rich analysis. Digital Humanities is ideally suited to make novel contributions to this under-conceptualized domain.

This forensic description of *Agrippa* is the result of an online cracking challenge that I created to marshal the expertise needed to crack *Agrippa*. The result of this challenge was that *Agrippa* was successfully reverse-engineered and tools were implemented to extract the ciphertext, crack it, and display the original plaintext. This interactive project is hosted online at http://crackingagrippa.net .

Forensic description of Agrippa

When *Agrippa* was first published there was considerable ambiguity about the poem's "mechanism." Some had suggested that it was a destructive virus, or that it triggered automatically when the disk was inserted into a computer. When Liu and Kirschenbaum began their forensic investigation of the software, they discovered that it was relatively easy to make a bit-for-bit copy of the disk using modern tools (Linux's dd). While no virus or automatic triggering was found, the team discovered that, as anticipated, the program would run only a single time. Of course, with a digital copy from the pristine original disk, infinite copies could be made. The program could be run time and time again, simply throwing away the "destroyed" version after each run. Based on information in the archival documents, Liu and Kirschenbaum assumed that the "self-destruct" mechanism was a (re-)encryption of the poem. While this guess turned out to be incorrect, running the program does result in a kind of digital *auto-da-fé* (it does "self-destruct") (Liu, 2004). Cryptography is central to the mechanism, but not as part of its self-destruction.

In conjunction with the interactive tools available online (as noted above), the following description corrects misconceptions about the technical workings of *Agrippa*. This exploration demonstrates that the cryptography in *Agrippa* was not used for destruction, but does contribute vitally to the aesthetic performance of *Agrippa.* Additionally, these interactive tools demonstrate that a round trip is possible: running the program results in a corrupted binary, but the ciphertext can still be extracted and cryptanalyzed, and consequently returning the original poem.

Even though the cryptographic algorithm turned out to be very insecure, even for its release in 1992, I quickly discovered that cracking *Agrippa* was a considerable technical challenge. Of course, without the prior efforts of *The Agrippa Files,* cracking *Agrippa* would have been a non-starter, since the obscurity of physical copies meant that there was no readily available binary before the creation of *The Agrippa Files*. But, even with the archival documents and the binary, after several weeks attempting cryptanalysis, I realized that I would need to enlist outside help.

I decided that I would marshal help by creating an online "cracking challenge." Cracking challenges are relatively common in some subcultures on the Web, but this one was complicated by the fact that *Agrippa* had been developed 20 years prior and seemed to follow very few industry practices. Cracking *Agrippa* requires knowledge of 1992-era Macintosh software development processes, tools, and languages. And once the software yields the ciphertext, the would-be cracker must possess skills of cryptanalysis.

Once the challenge website was launched – strategically tied to the cyberpunk ethos of William Gibson – interest in the project was considerable. I knew that my best chance of success was to tap into the hacker community, so I used social media to advertise the project narrowly. Within 12 hours, one of my advertisements was picked up by Cory

Doctorow at the blog Boing Boing, and with its massive readership, many other large technology-focused websites also began to spread the word. Eventually the locus of press activity crossed from technology sub-cultures to the mainstream. While the mainstream press brought many curious eyes, my website statistics revealed that very few people had the necessary skills (not to mention the inclination) to make an attempt. In the end, five submissions were received, all of which were deemed "successful" a loose criteria that crystallized around a working re-implementation of the cryptographic routines.

From the cracking challenge, I learned a couple of lessons in digital humanities project management, especially with respect to public-facing projects. I discovered that design and imagery matters, even if the time invested is not for your target audience. A tiny fraction of the website visitors stayed for longer than 30 seconds, most just long enough to send off a link to Twitter. Yet, this groundswell of activity was necessary to lure the tiny fraction that did possess the skill and inclination to contribute. In a similar vein, it pays to set up the project properly, providing all the necessary information and making it as easy as possible for people to engage. Finally, contests and other processes of gamification are typically utilized by the marketing industry but rarely employed for academic projects. Setting up a project in such a way requires a light touch, especially with respect to ethical concerns, but the outcome can be both beneficial to the researcher and fun for the participant.

After carefully working through the submissions with the contestants (who provided the substance of this account), I discovered that there are four main aspects to the *Agrippa* program: the compiled binary, the main cryptographic algorithm, the encryption effect that runs after the poem finishes scrolling, and the "self-destruct" mechanism that prohibits running the program more than once.

The compiled binary

The *Agrippa* program was developed using Macintosh Allegro Common Lisp, possibly version 1.2.2 or 1.2.3, and bundled as a self-extracting binary. As mentioned in *The Agrippa Files* archival documents, the initial plan for an auto-run, virus mechanism was never developed (Kirschenbaum, Reside, & Liu, 2008), and while there are some tricks to impede reverse-engineering the program, the programmer's boasts that it would be impossible to run through a debugger were unfounded. The Macintosh Lisp compiler uses Lempel-Ziv-Welch (LZW) compression, with the poem stored as encrypted text in a string variable ("zi") (one contest submission suggested that variable names may correspond to non-English words, in this case, "zi" means "words" in Chinese). This variable is encoded in the MacRoman character set, but only uses ASCII characters (low in the MacRoman table), so the visible effect is indistinguishable from ASCII. Offset values were discovered for most aspects of the *Agrippa* binary.

As was known in 1992 and exploited by a number of the contestants, Macintosh Lisp contains an error (not binding a keyboard handler at a particular point) that allows one to drop out of the program and into a Lisp Run-Eval-Print-Loop (REPL) console. With access to the REPL, arbitrary code may be run and interaction with the variables and routines reveals much of the source contents (however, the REPL is of limited utility because many variables are uninitialized, so constants are incorrect). By exploiting this error, it was discovered that the shipped production code does not exactly match the archived (partial) source code printout from *The Agrippa Files*. This suggests that the archival document was from an earlier stage of development. Several routines either changed names or no longer exist, e.g., UN-WAYMUTE-IT, UN-PERMUTE-IT, and UN-ROLL-THE-TEXT (which is thought to correspond to UN-ROLL-ZI).

The cryptographic algorithm

The *Agrippa* poem is pre-encrypted and stored as a string variable in the program. The ciphertext is not visible in the binary due to the LZW compression. All contestants discovered that the cryptographic algorithm is a custom Rivest, Shanker, Aldemann (RSA) function that encrypts in three-character blocks, with additional "bit-scrambling" permutations (a kind of simple substitution cipher). Because the poem comes pre-encrypted, there is no encryption routine in the program: the program simply loads the ciphertext, decrypts it to memory, and then abandons the plaintext (still in memory). As proof of this, the same decryption routine can work on a "fresh" disk as well as previously-run "corrupted" disk (more on the "corruption" below); two contestants implemented a tool to decrypt from the compiled binary in either state (requiring reverse-engineering of the LZW compression).

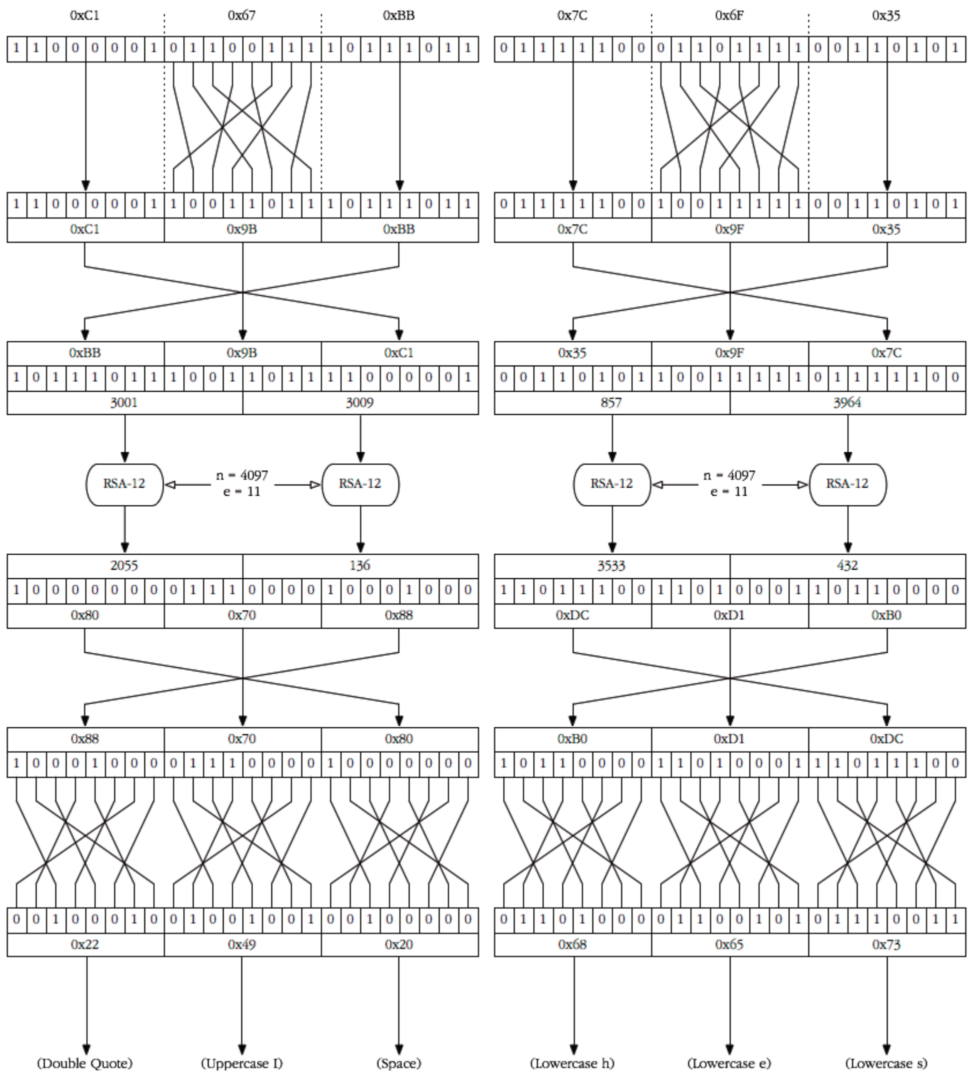
The cryptography is applied identically and independently to three-character blocks (resulting in cryptographic weaknesses; see below). On each block the cryptographic routine performs three distinct steps: first a bit permutation (substitution cipher) that converts three 8-bit characters into two 12-bit characters, then an RSA transformation using a 12-bit key, and finally another bit permutation that converts the two 12-bit characters back into three 8-bit characters (corresponding to ASCII-encoded text).

The RSA cryptography has a public modulus of 4,097 (from primes 17 x 241), and a public exponent of 11. Due to the extremely short bit-length of the public modulus and exponent, the private exponent can be found easily (either through brute force or the Chinese Remainder Theorem), and was revealed to be 3,491 (the private modulus is always the same as the public modulus, 4,097). So the RSA encryption process is to take some number x to the 3491st power, modulus 4,097; however, remember that the encryption routine is not present in *Agrippa*, and was reverse-engineered once the private exponent was discovered.

Evidently the anonymous *Agrippa* programmer was either undecided or confused about what kind of encryption to employ, remarking in a letter that the cryptography would be similar to both Data Encryption Standard (DES) and RSA (the prior being a symmetrical key cryptography algorithm, the latter asymmetrical): "another info source would be anything on the Data Encryption standard or mathematical works by Rivest, Shanker [*sic*], Aldemann [*sic*]" (Anonymous, 1992, p. 7). Likewise, the cryptographic routines are named with references to DES, even though they are RSA.

The anonymous programmer attempted to strengthen the cryptography by using a more sophisticated mechanism for block enciphering (where the same key is re-used for each block, but ideally "mixed" with the neighboring block). The programmer

Figure 1: Contestant Jeremy Cooper's graphical depiction of the decryption process.



Source: http://www.crackingagrippa.net/submissions/jeremy_cooper.html

remarked, "[t]he value, both character and numerical, of any particular character is determined by the characters next to it, which from a cryptoanalysis [*sic*] or code-breaking point of view is an utter nightmare" (Anonymous, 1992, p. 2). Yet in reality the encryption is applied identically to each three-character block, in a mode of operation known as Electronic Codebook (ECB). This simple mode of operation has many cryptographic weaknesses, most visibly the fact that identical blocks will encrypt to the same result. For example, in the plaintext of the poem, there are numerous sections of three consecutive spaces which encrypt to "space, e with circumflex, backslash," or in decimal 20 136 92 (with quotes added for clarity, displayed as ASCII: " ê\"). Even without reverse-engineering the algorithm, this weakness is significant and exposes the ciphertext to old-fashioned statistical analysis (where common three-letter words or three-letter combinations would be visible in the ciphertext). Similarly, the three-character block size and ECB mode of operation explain the curious two spaces at the end of the poem (Wiedijk, 2011). Rather than signalling the end of the poem as Wiedijk opined, the two spaces at the end of the poem are needed to pad the block, otherwise the cryptographic routine would fail.

When *Agrippa* was released in 1992, the United States famously classified all cryptographic materials as munitions and restricted the export of "strong" cryptography (Diffie & Landau, 2007). *Agrippa* was seen by cipherpunks and computer scientists as a challenge to these stifling and backwards cryptography export controls. Yet given the extremely short key-length (12 bits), *Agrippa* would never have been prevented from export – in 1992 the United States permitted an RSA modulus of 512 bits. Indeed, the political involvement of John Perry Barlow and the Electronic Frontier Foundation was superfluous given that *Agrippa* would not have been considered strong cryptography (Barlow, 1992).

Encryption effect

When *Agrippa* is run, the poem slowly scrolls down the screen, and once the poem has finished scrolling, it displays an encryption effect, seemingly to evoke the idea that the poem is re-encrypted. As discussed above, there is no RSA encryption algorithm in the binary; however, by running the plaintext through a permutation routine (re-purposed from the main *de*cryption algorithm), the plaintext is effectively encrypted using a simple substitution cipher. While this is a re-encryption of the plaintext, it is actually only for visual effect since the ciphertext generated by the substitution cipher is not saved back to disk, but is instead displayed and then abandoned. Once *Agrippa* is run, only one change is saved to disk: the "self-destruct" mechanism.

The "self-destruct" mechanism

Agrippa famously runs only a single time. There are a number of possible mechanisms to cause a self-modifying program to run only once. For example, one could flip a switch in the binary that alerts the main routine that the program has previously been run, or re-encrypt the data and throw the key away (possibly generated dynamically with runtime variables), and so on. The anonymous programmer of *Agrippa* chose a simple mechanism: write a large string of data over a portion of the binary that contains necessary run routines. In the archived source code printout, this self-destruct mechanism wrote 40,000 ASCII characters (ASCII code 255) to a specified offset, leaving a string of 320,000 binary 1's to corrupt the program. Evidently, at some later stage of development, someone thought it would be more in keeping with the theme to write a fake genetic sequence (CTAG's) instead of merely 1's.

This self-destruct routine is called MAKE-SOME-SHIT, and is located in the archival source code listing halfway across page three and the missing page four. One contestant speculated that page four of the source code listing might have been omitted from the *Agrippa Files* archive due to the presence of the word "SHIT" in the routine name. It was revealed that MAKE-SOME-SHIT uses a fixed seed to call the Mac Toolbox Random Number Generator, which saves 6,000 characters (either C, T, A, or G) to the disk (at offset 680168). While the offset chosen for self-destruction does effectively corrupt the program, it does not destroy the ciphertext. Two cryptanalysis implementations can decrypt the poem from either a "fresh" or corrupted binary, since the self-destruct mechanism left the ciphertext intact.

Conclusion

Cracking the *Agrippa* code is yet another performance of this fascinating artifact. I showed that while Liu's argument about the "destructive creativity" of encryption is incorrect, something of equal fascination has taken its place. Additionally, Kirschenbaum's argument that there is "no round trip" is mistaken, but this mistake suggests that much work is needed to better conceptualize cryptography. By revealing the mechanism in *Agrippa*, I showed how cryptography does round trip, laying the groundwork for a more philosophically robust re-conceptualization of cryptography (to be taken up elsewhere). I suggested that multidisciplinary or even public collaborations are a fruitful method for digital humanists interested in interrogating complex technical objects. As is often the case, the success of the cracking challenge was due in large part to the positioning and "marketing" of the project, piquing interest and generating a groundswell of activity. I have shown that the interpretation of cryptography and other code systems should become a matter for digital humanists, especially those in critical code studies and software studies (Fuller, 2008).

References

Anonymous. (1992, March 28). Letter from Programmer (Item #D6) (transcription). *The Agrippa Files*. URL: http://agrippa.english.ucsb.edu/letter-from-programmer-item-d6-transcription [October 18, 2012].

Barlow, J.P. (1992, June 11). Letter from John Perry Barlow to Kevin Begos (Item #D45) (transcription). *The Agrippa Files*. http://agrippa.english.ucsb.edu/letter-from-john-perry-barlow-to-kevin-begos-11-june-1992-itemd45-transcription [October 19, 2012].

Diffie, W., & Landau, S. (2007). The export of cryptography in the 20th and the 21st centuries. In Karl de Leeuw & Jan Bergstra, (Eds.), *The history of information security: A comprehensive handbook.* Elsevier, pp. 725-736.

Fuller, M. (2008). *Software studies: A lexicon*. Cambridge, MA: MIT Press.

Kirschenbaum, M.G. (2008). *Mechanisms: New media and the forensic imagination*. Cambridge, MA: MIT Press.

Kirschenbaum, M.G., Reside, D., & Liu A. (2008). No Round Trip: Two New Primary Sources for *Agrippa*. URL: http://agrippa.english.ucsb.edu/kirschenbaum-matthew-g-with-doug-reside-and-alan-liu-no-round-trip-two-new-primary-sources-for-agrippa [October 19, 2012].

Latour, B. (1988). *Science in action: How to follow scientists and engineers through society* (REP.) Cambridge, MA: Harvard University Press.

Liu, A. (2004). *The laws of cool: Knowledge work and the culture of information*. Chicago, IL: University of Chicago Press.

Liu, A., Hehmeyer, P., Hodge, J., Knight, K., Roh, D., Swanstrom, E., & Kirschenbaum, M. (2008). *The Agrippa Files.* URL: http://agrippa.english.ucsb.edu/ [October 18, 2012].

Shannon, C. E. (1949). Communication theory of secrecy systems. *Bell System Technical Journal*, *28*(4), 656-715.

Wiedijk, F. (2011, July 17). Original text of Gibson's "Agrippa" poem extracted from disk. URL: http://agrippa.english.ucsb.edu/post/documents-subcategories/the-disk-and-its-code/original-text-of-gibsons-agrippa-poem-extracted-from-disk [October 19, 2012].