Authorship
infinite authentication tool

Computational difficulty and super-flexibility in endless cryptographic evidence
(publicly-verifiable authorized-only number modification)

Proof of life

User authentication

Canary update authentication

Data, author, and version control authentication

Unretentive suspend-able untraceable anonymous cryptocurrency
(publicly-verifiable authorized-only withdrawal or ownership hand-over)

Untraceable anonymous fair trade over any network
(cryptographic solution to the Prisoner's Dilemma)

Group and military authorization

Animal theft management

Voter fraud prevention

-Includes $P \neq NP$ proof-

Introduction

Authorship allows you to generate a special number which can **express any personal message.** You can modify this number and continue including new messages without anyone else being able to modify it for you—**"authorized only."** At the same time, the public is given the mathematical ability to verify that your modifications are numerically correct —**"publicly-verifiable."** The combination of these three properties means the public or verifying party can rest assured that your messages are not impersonation as you continue to address the public as often as you wish—**"infinity."** Any Authorship methods and components can be swapped and layered at any time meaning you are never locked in to the protocol—**"super-flexibility."** And with the use of all-way step-down functions curated for Authorship as described in the included P≠NP proof, Authorship guarantees that no reversal shortcuts exist to its multi-way functions, meaning all cryptanalysis necessarily begins with trial strings—**"computational difficulty."**

You are expected to publish your first number before you are subject to censorship or worse. Once you become a target, the public will inquire about your well-being. You are then to release a new number so the public can verify your authentication event by testing the relationship between your new number and the number before that. Verifying parties may now overwrite your old number with the new one, as well as discard any and all old modification information. Essentially, if you trust the ownership of some first number, you can trust that its evolution from there is authorized explicitly. The following options are presented in the terminal Command Line Interface.

1 Get Authorship number   (Generates your first number. This option is used once unless you know what you're doing.)
2 Modify Authorship number   (Generates your new number. Your old number is then modified to represent this new one, as well as any message of your choice.)
3 Verify number modification (Generates their new number & message using their published file. This file's function-compression must match their old number.)

Glass vault analogy – Authorship numbers are comparable to a transparent vault meant to sit out in public view. Those who watched you install your vault can trust that it's yours. Any data displayed inside the vault is open to observation yet only you can manipulate that data. The best part is—your disposable random keys are selectively published for every authentication event and never seen or utilized again. And purposely unused keys are destroyed immediately. In technical terms, when you change something in your vault, you actually build a new vault just before that, then shatter the old one in such a way so as to point each fragment at the new vault—leaving an exact numerical representation of the new vault fingerprint.

Proof of life

This ability is useful when the user has become the subject of censorship or worse. The user can—especially upon request—assure the public or verifying party that any further correspondence is not impersonation. "Prove that you're alive, for as long as you live."

User authentication

Public key cryptography lacks strong authentication, and any other system can be fooled using clever and often processor-intensive computation. Authorship allows you to prove that you are who you say you are over the internet and this system can be intertwined with any key exchange or messaging tools.

Canary update authentication

A canary is a message updated monthly by some party or service on the internet. For every new canary message, the service user-base may rest assured that no abusive power demanded user data, otherwise canary updates terminate. Authorship can ensure that no canary is updated without special keys, even if—as seen before—such abusive powers seize personal and company property.

Data, author, and version control authentication

For every authentication event, the user may insert any message, text, or data such as a piece of work or its hash. Those relying on cryptographic tools may need to verify that some work was written by or updated by the trusted author of their choice. Authorship allows any author and those maintaining software to authenticate releases and version control. Abusive powers have admitted to modifying and undermining cryptographic systems not of their own work.

Unretentive suspend-able untraceable anonymous cryptocurrency

This system is not sensitive to heavy computation or timely release. Authorship-based circulation can be suspended and used again after arbitrary periods of time. During transaction inactivity the system is dead silent, after which nothing needs adjusting. Public verification simply picks up where it left off. A circulation can be as small as having two users. No identification or accounts, decentralized, nothing to mine. Numbers cannot be traced to any person as they only appear in public while their modifications are released anonymously. Keys to Authorship numbers are always generated privately, it's their ownership that's handed over. Otherwise the owner can simply authorize a deduction from their number's worth—something assigned to the numbers upon circulation initialization. And you may discard any and all old transaction and modification information after verifying some event. See page 7 for details and the anonymous initialization of any public, exclusive, and private circulation.

Untraceable anonymous fair trade over any network

This is a direct solution to the Prisoner's Dilemma. Here, you have something of numerical value to trade in the first place—the ownership to Authorship numbers. Alice and Bob can trade number ownership where trade is always either fair or nonexistent. Anything assigned to these numbers would swap hands whether it's a boat, club membership, or military authorization capabilities for events in the future. This applies over any network, meaning Alice can give up her end of the deal while Bob might be away or offline at the moment. If Bob decides to cooperate with Alice when he gets the chance, the trade is publicly-verifiable. And if someone defects, the one who gave up their end of the deal reverts to normal number modification as if nothing happened. Whatever it is assigned to these numbers remains in the control of their owner if there is any defection in trade—at the cost of little wasted computation. See page 8 for detailed documentation.

Prerequisite concepts

(Contents: lossless versus lossy compression, deductive lossy compression, cheating limits with external information, hash functions, assigning values to things that don't exist, one-way, multi-way, and all-way functions, stepping down from all-way to multi-way functions as proof of the existence of computational difficulty, and multi-way functions highly curated for publicly-verifiable authorized-only systems while retaining the impossibility of any reversal shortcuts.)
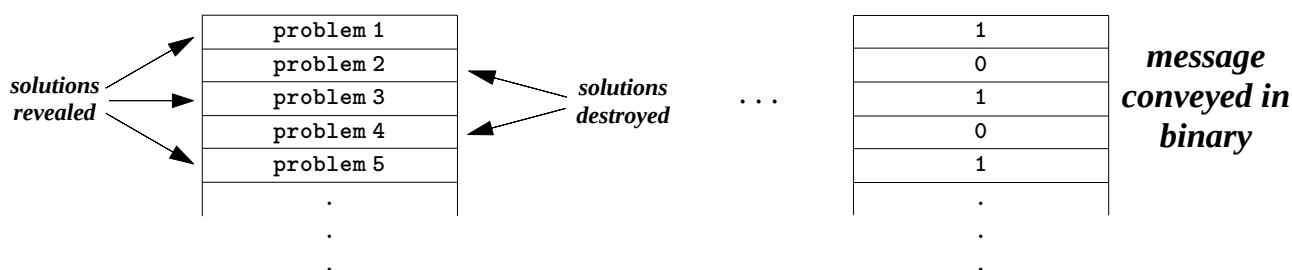
1. Infinitely many things are pointing at x – You can say that compression algorithms are the external information that comes with every compressed file. Lossless algorithms decompress files where the original file is reproduced exactly. Lossy compression on the other hand, produces smaller and purposely degraded files without completely depleting their usefulness. A good portion of information in lossy compression is lost and it could be a range of values, pixels, and even physical matter. Without access to the original file, it is impossible to know the exact contents of that file. Consider sharing the raw file along with its lossy-compressed version. The two files appear to be related, however, anyone can make tiny selective changes to the raw file and say their version of the raw file is the one intended to be related to the lossy version. This would be indisputable. If some image pixels didn't make it through the compression, then depending on the algorithm, those pixels could have been any color within some plausible unintended raw file produced by the attacker.

This is where the hash function comes in useful. Hash functions are highly lossy compression algorithms who make good use of the avalanche effect. Each symbol in a raw input file dramatically changes the output lossy version. When the process is finished, the output should look like a random number. Now, if a slightly modified raw file is fed to the same algorithm, the output produced would be dramatically different compared to the output of the original. You can logically deduce and with great deal of certainty that only a certain raw file had produced the hash in question. By definition, hash functions process data quickly. Authorship uses the avalanche effect strictly for near-lossless compression while processing data slower than would a hash engine hence the name "deductive lossy compression." And depending on the engine, infinitely many things or raw files can produce the same hash—unless you set engine input constraints and expectations of raw files with identifying properties—as done in the Authorship deductive lossy compression.

2. Generating difficult questions in bulk – Conclusions in philosophical complexity often times require question reformulation or surprisingly distant external information. Complexities arise when you discover that two very distant thoughts are pointing at one another. Then, anyone attempting to know what you know must either try all possible thought combinations or come up with some shortcut that takes them to where you have been. This is the essence of cryptography where information is not always numerical.

Cryptographic algorithms are built for generating difficulties by first generating massive random solutions then constructing functions who utilize and obfuscate those solutions. This is similar to maze construction where you begin with a functional pathway then add faulty distractions. Authorship generates large quantities of multi-way functions for its difficult questions. Deductive lossy compression of such a list will establish the function order in that list and serves as a near-lossless compression since the raw list is provided for decompression.

3. Missing information is information – You can assign values to any expected items not present for some event, then include those values in your description of that event. This way, the host of such an event conveys a message that is unambiguous if you were told to expect a certain number of missing items. For some event where items appear in order, the fixed fraction of any missing items tells a bigger story. Here, "solutions to difficult problems" are the present and missing items. Present = 1, missing = 0. This symbolism is a binary code extracted from the event. And having already published an ordered and compressed list of difficult problems, you can fabricate any new information publicly-known to have came from you since only you posses solutions to the problem list. And the problems in any list can be something as simple as encrypted files or some other difficulties to which only you have the answer key.

solutions revealed →	problem 1		1	*message conveyed in binary*
	problem 2		0	
	problem 3	← solutions destroyed	1	
	problem 4		0	
	problem 5		1	
	.	. . .	.	
	.		.	
	.		.	

The proof of the existence of perfect secrecy through all-way functions and computational difficulty through multi-way functions in cryptography

Perfect secrecy through all-way functions

The One-time pad offers fast all-way function based encryption with unlimited plausible deniability and guaranteed privacy. All-way functions are those with maximum reversal solutions not exceeding deductive reasoning given the key. Any guess key successfully decrypts all-way function based files and each guess key produces a different output making it impossible to say which output was intended by the key owner for that file. Brute force attacks produce plaintext with every possible symbol combination for that length. The following tables demonstrate all-way nature using modular arithmetic.

Ten-symbol addition possibilities of (a+b) mod 10

There are ten of each digit in the grid. Every column and row contains one of each digit. Every digit yields ten different digits. If only one value is known, the other two are unknown. If two values are known, the third can be logically deduced. There are 10^2 total outputs.

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

Three-symbol addition possibilities

Similar rules apply for any number of symbols. Here, for every output symbol, three possible input pairs exist and this does not exceed deductive reasoning. For every guess key symbol, only one plaintext symbol is made responsible and for every guess plaintext symbol, only one key symbol is made responsible.

	plaintext		
key symbol	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Encryption sample for ten symbols

The output stream borrows random information from the key therefore repetitive data does not degrade file randomness as shown where the plaintext repeats excessively.

random disposable key = | 7 | 2 | 3 | 8 | 7 | 2 | 5 | 3 | 7 | 4 | 8 | 1 | 6 | 9 | 3 | 3 | 2 | 7 | 0 | 9 | 9 | 2 | 4 | 1 | 4 | 1 | 6 | 0 | 2 | 2 |

plaintext in base ten = | 7 | 9 | 2 | 3 | 4 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 4 | 5 | 0 | 5 | 3 | 7 | 4 | 9 |

ciphertext output = | 4 | 1 | 5 | 1 | 1 | 4 | 7 | 5 | 9 | 6 | 0 | 3 | 8 | 1 | 5 | 3 | 2 | 7 | 0 | 9 | 9 | 1 | 8 | 6 | 4 | 6 | 9 | 7 | 4 | 1 |

⇧

$(7+7) \bmod 10 = 4$

Privacy is guaranteed if random disposable keys are first shared in private. Users planning to share large files must first share significantly large key files in private. Whatever the stored or transmitted encrypted data, its contents cannot be related to you if your copy of the keys is destroyed. And anyone with keys who decrypt your data into something plausible or even illegal still cannot prove the intended output of your data.

Ten-symbol systems are said to have a plausibility ratio of 1:10 and three-symbol systems—a plausibility ratio of 1:3. This ratio may approach infinity and is calculated using the formula: 1 to number of symbols—or to be mathematical—number of symbols to number of symbols squared. Higher ratios compensate for weak random number generation.

The encryption sample for ten symbols demonstrates obfuscation of the fixed-length plaintext of 30 digits. The message itself may be less than or equal to 30 digits since any remaining space is dynamically filled with new random numbers. Had the key length been adjusted to match every plaintext length, the adversary would have a hint at what a short message could be. And had there been no random numbers appended to each message, key digits who drop to fill the remaining output space would give the cryptanalyst a closer look at the exposed key fragments and their generating algorithm.

Computational difficulty through multi-way functions

So far, perfect secrecy has been shown to exist for all-way functions where privately shared keys are used only once and demolished effectively. As the user runs out of keys, such inconvenient protocols require that more keys are shared necessarily in private. Now consider reusing and transforming an existing key or set of keys rather than sharing new ones in private again. This is the immediate step down from the all-way function. Individual output symbols continue with having maximum reversal solutions not exceeding logical deduction, however, with weakness and order introduced to the keys and with more user data to sample—plausible solution possibilities only narrow.

Given only encrypted output as shown in the encryption sample for ten symbols, guessing not only the initial key but the key transformations based on what some algorithm thinks decryption might look like for ongoing messages is a difficult task indeed. Due to the now limited plausible deniability, there still cannot exist reversal shortcuts who compute the intended output for each message therefore search work is inevitable.

With the preferred convenience of one-time privately shared keys, using this proof multi-way function based encryption with key transformation may now be considered a protocol for which decryption difficulty is guaranteed. A key transformed dynamically and symmetrically on both ends in secret can protect large amounts of data before the key transformation details are guessed correctly, if ever. This proof describes a problem classifiable under NP (easily verifiable solutions) but not under P (easily discoverable solutions) therefore P≠NP.

1. All-way functions – Solutions to all-way functions all come to the same conclusion. Each symbol obfuscates a number of solutions equal to the number of different symbols. The number of possible keys to any such symbol or string of symbols is equal to (number of different symbols) raised to the power of (total number of symbols.) Given a disposable key, which solution was intended by the key is the question without answers. Reusing or transforming that key, however, provides a point of entry for the existence of key-data relationships.

2. Multi-way functions – As with all-way functions, solutions to multi-way functions all come to the same conclusion. For example: the hash to which multiple expected strings purposely correspond, where one of those strings may be the intended message or key. And one string solution does not necessarily reveal another. There may or may not be a shortcut to each one of those solutions, however, no shortcut exists for the one intended solution where keys are transformed, and the message is represented using multiple symbols strung together (see Encryption sample for ten digits.) Intended solutions are defined by the generated key and its ongoing transformations. On the other hand one-way functions are not born intertwined and normally do not contain other plausible information on their own.

3. P≠NP is both proven and unproven –
P≠NP is unproven – In one reality, if ignoring key transformation and without cryptanalysis, any values tried for these encrypted file types cannot be mathematically rejected. Conventional verification algorithms not curated for cryptanalysis may consider these trial values to be solutions, however, that is not the stated problem. And it is possible —though difficult—to answer the stated problem…
P≠NP is proven – In the other reality, cryptanalytic search algorithms successfully reject key transformations responsible for random plaintext. Over time and with more user data to sample, the algorithm reveals a transformation responsible for genuine non-random user data, after which any new user messages are successfully decrypted. Such an algorithm treats the multi-way function as a one-way function based on its search priorities. And no reversal shortcuts exist as only through heavy search work it is possible to determine the corresponding shared secret.

4. The cryptography community is satisfied either way – P versus NP is only a question inquiring about—among other things—information security, the possibilities and impossibilities of crunching large numbers, and whether slightly longer ciphertext poses exponentially greater difficulties. Whether or not you choose that P≠NP has been proven, cryptographers can finally rest assured that perfect secrecy and computational difficulty is possible and here to stay!

5. One-way functions are probably out of the question – Decryption difficulty has been proven to exist by first proving the existence of perfect secrecy, then taking steps down from there. This is comparable to becoming aware of the farthest point of attack, then navigating away and arriving at the solution while having gone through an impractical number of steps. And you could not have determined the usefulness of one step without first having tested the step before that and so on.

Consider looking for shortcuts to some one-way function given correct solutions. Take for example at least two unique semiprimes of cryptographic strength. The objective is to build a single engine who can transform necessarily both semiprimes into their constituent primes, one semiprime at a time, and without using division. This can be tried in reverse where the objective is to transform the four primes into their products without using multiplication. This algorithm's reverse reformulation would then factor semiprimes quickly. Any successful engine is then fed a significant quantity of semiprimes for those who need some assurance of functionality. And there may be multiple engines who do the same thing, however, their function cannot be classified under P or NP unless the proof of reversal shortcut functionality is constructed—at least for some interval, set, or special cases.

It is possible to begin with solutions when looking for shortcuts to those solutions, but where are the boundaries of useful information and is there anything of interest in between? Without an awareness of such a landscape it seems the search is a computational difficulty of its own. See #2 (multi-way functions) for the reasoning behind the exclusion of multi-way functions in this conjecture. Without key transformation, even multi-way functions can achieve perfect secrecy by assigning each key to some complete list of symbols where in comparison to the function's decrypted output, the intended key weight represents the order of its assigned symbol.

6. <u>**Computational difficulty in multi-way functions curated for publicly-verifiable authorized-only systems**</u> –
This point should help visualize computational difficulty and build confidence in raw or layered multi-way functions in CSC (Cryptographic Semiprime Coupling) and its superior with computational difficulty—Authorship. The objective is to construct concise multi-way functions to which one intended key and many-to-none unintentional plausible keys correspond—where the discovery difficulty of any key is strong and roughly equal. Although a single key is preferable, cryptanalytic search priorities for plaintext may successfully verify unintended keys due to the nature of limited plausible deniability as solution possibilities narrow with key reuse or transformation. The following sample multi-way function significantly reduces the verifiable key quantity while retaining the impossibility of any reversal shortcuts based on some built-in search priorities meant for public distribution.
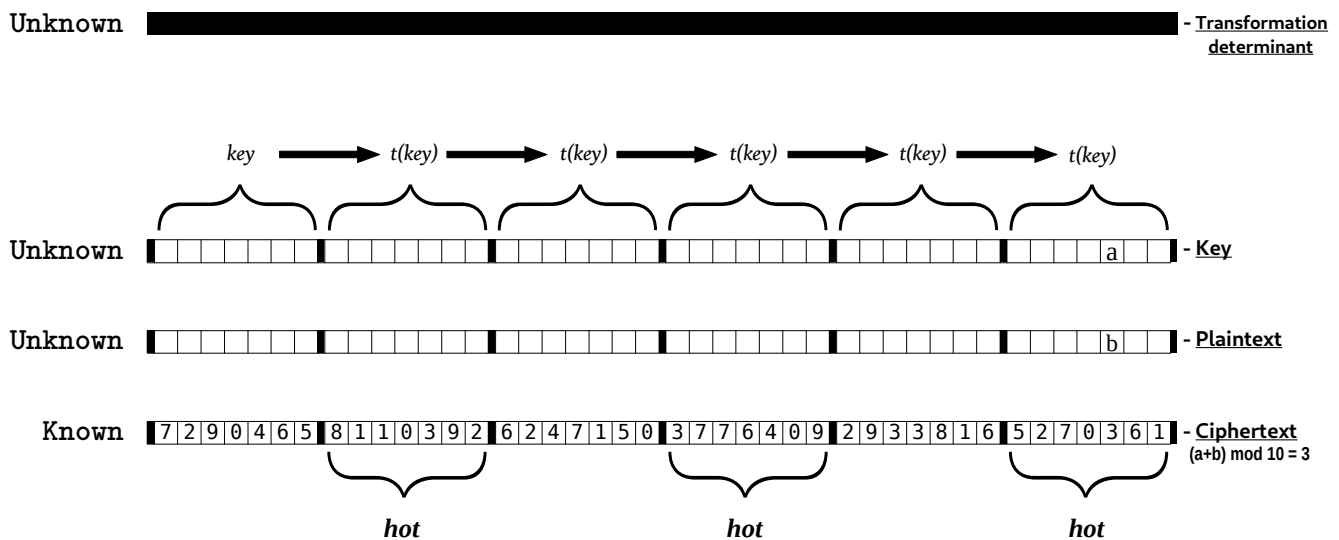
Transformation determinant: only the input values are random. They are the transformation guide in some built-in protocol. **Key:** only the first sub-key is random. Its transformations are predeterminate from there. **Plaintext:** strings can be identified and verified for their symbolic or numerical properties. **Ciphertext:** multi-way function with sub-key to sub-function order correspondence.

Cool sub-functions obfuscate random plaintext. Hot sub-functions obfuscate plaintext with identifying properties expected by the search priorities. For example, each sub-plaintext in a hot zone must be composed of contiguous primes of any length. Here, public decryption may include a string of digits each representing the prime digit length and in order—so as to avoid tree search, ambiguity, and the identification of prime sub-strings within the primes. Zone parameters and the transformation type would be built-in and distributed—along with search priorities.

Whatever the identifying plaintext properties required to pass verification, neighboring sub-functions in the cool zones show unlimited plausible deniability through random output. However, the presence of complete data samples, general search priorities, and predeterminate key transformation establishes the strong relationship between the transformation determinant, key, and plaintext—in congruity with ciphertext open to observation.

Now the function dramatically steps down to multi-way whose cryptanalysis necessarily begins with trial strings in order to determine their transformed usefulness in the neighboring sub-functions hence the persistent difficulty in discovering intended solutions and plausible additional solutions if any—based on corresponding search priority.

Modifying Authorship numbers

Authorship numbers are the hash or deductive lossy compression of a long list of multi-way functions, locked into their order in that list due to the list being compressed sequentially as a whole. Multi-way functions are generated first then compressed to form the Authorship number. Recall that you can use this ordered list to convey any message by selectively revealing solutions to the list functions.

Now consider generating a new number, *then* forcing your old list to describe this new number exactly—by selectively revealing solutions to that old list. Solution revealed = 1, solution missing = 0. This binary code can represent your new number. Anyone verifying your authentication event will overwrite your old number with your new one then discard any old modification information. As long as you generate a new slate of functions and describe them all with some number of yours, you can continue authentication events indefinitely.

Not all functions in the list are reserved for describing new numbers. There is a partition reserved for the user's message. This is how the public or verifying party extracts your messages symbol for symbol. The ability to insert any information not yet fabricated eliminates the need to peer into the future or resort to the one-time precomputation of a single finite chain of sequential Authorship numbers.
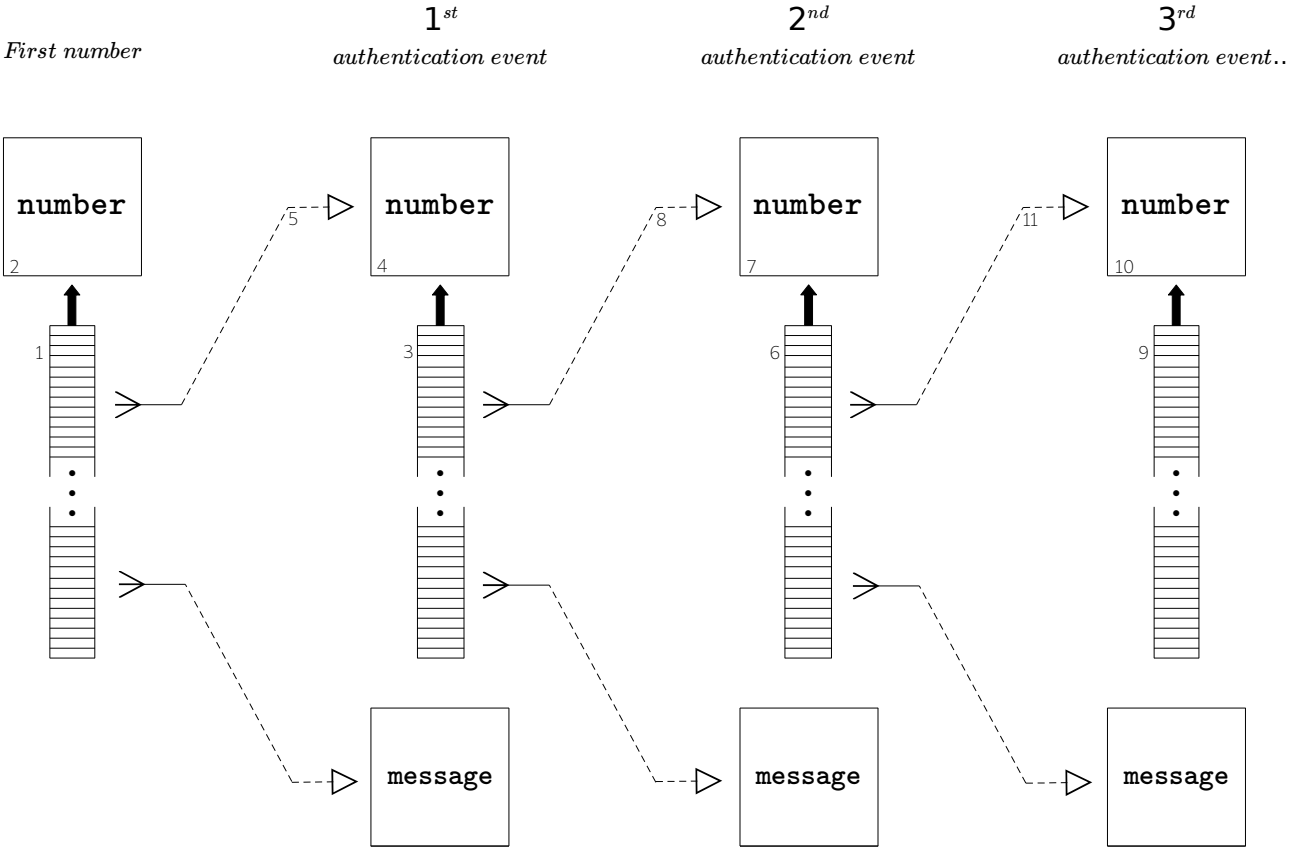
For every nine contiguous multi-way functions in a list, only five are utilized. In the sample list (100001111) the "ones" represent problem solutions who were published while "zeros" represent problems for which solutions have been permanently destroyed by the user after modifying their number. This combination of ones and zeros represent the first two digits of the new number. Had the binary representation been chronological rather than requiring a fixed fraction of ones, censorship of published solutions would give false modification information— possibly benefiting the attacker. Instead, Authorship expects exactly five solutions out of every nine contiguous problems before that modification information gives any meaning. See page 9 for details on symbol assignment.

Operation principles

1. <u>Authorship numbers</u> – All Authorship numbers are 1,000 digits long, not including the user's message. The public must retain only your latest number in order to continue verifying its modification. Authorship numbers are the compressed output of 13,500 multi-way functions. Each list of functions points to the next list *compression* (number) rather than the entire list itself since it would be impossible to precisely represent a new list of 13,500 functions, each 12,000 digits long—using only 13,500 functions who give "1" if solved and "0" if left unsolved. Additionally, the compression gains strength in that the input is not only fixed but is expected to be filled with multi-way functions.

2. <u>Authorship functions</u> – Refer to the proof of the existence of computational difficulty (pages 3 to 5) and review #6. Authorship multi-way functions are 12,000 digits long with the same six partitions, each 2,000 digits. Each hot zone is composed of some fluctuating quantity of primes ranging from single-digit to five-digit primes. This averages out to 535 contiguous primes for each sub-function in a hot zone—of which there are three per multi-way function.

And again, <u>solution revelation includes three strings of digits representing prime length in order, as well as the first sub-key and the transformation *determinant.*</u> It is convenient to simply call these ingredients "key." Plaintext can be deduced using the known ciphertext and the keys produced. Primes are generated randomly and are used as the expected identifying properties for public verification.

3. <u>All multi-way functions are random and disposable</u> – All solution ingredients are random, used only once, exposed after use, and never utilized or seen again. Purposely unused solution ingredients are immediately destroyed. This means functions meant never to be solved remain unsolved and convey the binary symbol 0.

4. <u>Cryptographer's acknowledgment</u> – You may need to verify that your new number is available to the public or verifying party—as you can be censored in some extreme circumstances. Here, the verifying party may arrange to modify their number in response and you can rest assured that your authentication event has been acknowledged.

5. <u>Cryptosabotage is futile</u> – Due to the large number of solution ingredients required to modify an Authorship number, in the unlikely event that a few or even a few thousand multi-way functions are solved by some attacker, you may continue normal operations while ignoring the unfit modification information.

6. <u>Multi-way functions with regulated solution quantities</u> – Number modification needs not necessarily binary-based representation. You can have some multi-way function whose multiple solutions represent a single digit—per solution. This would help save on space and computation time compared to the use of nine whole Authorship functions just to represent a digit pair. One example function is the hash to which multiple expected strings purposely correspond.

7. <u>Infinity and super-flexibility without breaking trust</u> – For any transaction or authentication event, your user message may one day include the source code hash to some Authorship variation, and a declaration of the continuation of number modification using the program of your choice. You may hot-swap to or insert another number style, method, or function of any type or size at any time and continue authentication events using the new modification system while retaining the same public trust given to your first number. Meanwhile, the original Authorship number may continue running alongside the new system for redundancy or backup is needed. Not only can you fork and generate alternate paths, but you can insert any simple functions such as a hash, encrypted files, or other difficulties and impractical items.

You can hot-swap to longer functions at any time therefore—in many ways—authentication events may approach infinity. Additionally, amid number modification, the Authorship protocol and its variations can be layered indefinitely without authentication interference. Thanks to the data insertion capabilities of Authorship, users are never locked into the protocol. Initially, Authorship provides a point of entry for endless cryptographic evidence.

8. <u>Modification as a cryptocurrency: publicly-verifiable authorized-only withdrawal or ownership hand-over</u> -
Authorship can be used as a cryptocurrency—as is. Consider the initialization. All participants retain a list of all the Authorship numbers in some exclusive circulation. Those who generate whatever number own its keys and therefore own the number. In this example, all participants agree on a finite quantity of indivisible Authorship numbers whose monetary values are immutable and do not fluctuate.

Receiving an Authorship number is accomplished in two simple steps. First—privately generate a new number as you normally would, and send it anonymously to the other party. Second—they force one of their numbers to point at your new number and anonymously publish that modification information. Essentially their number's ownership is destroyed and reincarnated privately in your device which now corresponds to your keys only. All other users can now verify the ownership hand-over and overwrite the old number with the new one in their retained list of all available packages. It is unknown who that number came from or to whom it now belongs but its value has been teleported to someone who privately generated its new random disposable keys, and only that new owner can authorize its next hand-over. Any and all old modification information can be discarded after verification.

Now let's say each user gets only one Authorship number. Here, some initial dollar value can be assigned to each number for the circulation initialization. Sending digital money or adjusting number worth is as simple as expressing the message within your modification: "devalue here by $5 then add 5.00 to Authorship number 3174…..4011." And again, old transaction and modification information can be discarded after verification. As you can see, given the properties of data insertion, permission, and verification, Authorship has endless applications. For a highly modular example—multi-user authorization—a system of number assembly utilizing functions from different users. Here, authentication events are verifiable if and only if all participants are satisfied and selectively release their keys to the single number.

# Untraceable anonymous fair trade over any network
(cryptographic solution to the Prisoner's Dilemma)

Prerequisite documentation: Authorship or CSC (Cryptographic Semiprime Coupling.)
Recommended research: exhaustive simulations of the Prisoner's Dilemma
and its reach in philosophy, international politics, and everyday life.

Suppose Alice and Bob wish to swap number ownership over the internet—Alice wants his number and Bob wants hers. Anything assigned to the Authorship numbers involved would swap hands whether it's monetary value, highly prized property, secret membership, or military authorization capabilities for events in the future. Recall that it is already possible to simply hand over Authorship numbers by representing a number privately generated by someone anonymous. From there, the only problem would be whether or not—amid sharing over some network—the other party had honestly handed their number over to you. Using Authorship, Alice and Bob can trade one-for-one, two-for-two, or any combination by specifying the trade information and modifying each and every number involved.

The solution is not dependent on the chance of equal cooperation. No matter who uploaded first or held more number fragments at any point in time, trade is always either fair or nonexistent. And there is never loss on either end. Numbers are given backup paths just before the trade. Fair trade devalues the backups while defection devalues the trade.

If trade is fair, backup paths are devalued and both Alice and Bob now modify a number to which were assigned the valuable goods they were each looking to acquire.

Alternate modification paths
are created first on both ends.
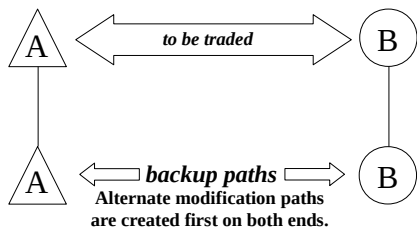
If there were any malicious defects in trade or if one party changes their mind at the very last moment, the traded items are devalued and both Alice and Bob continue modifying their backup paths as if nothing happened. Any goods assigned to these numbers remain assigned while the numbers simply change with no effect.
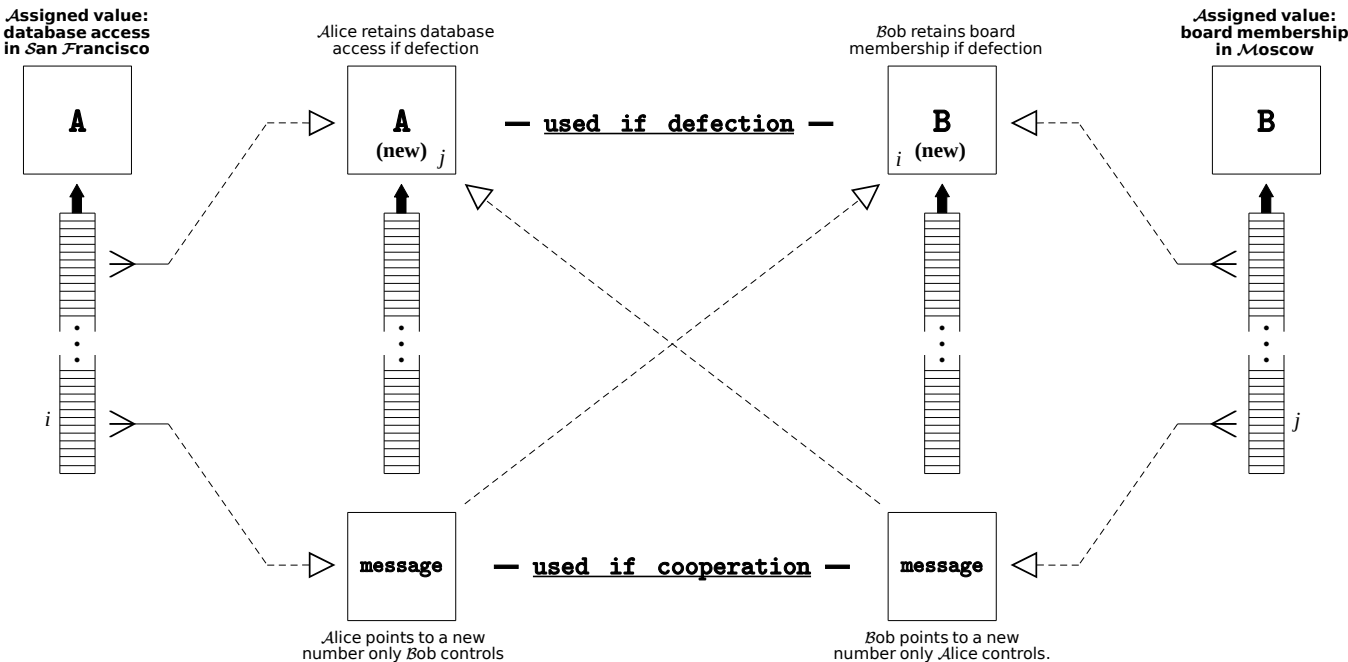
Alice *Bob*

Using only the function list partition reserved for representing Authorship numbers, both Alice and Bob modify their number the usual way. This gives them an alternate modification path.	
Alice sends her new number to Bob.	**Bob sends his new number to Alice.**
Alice's user-message for this event now includes Bob's new number.	**Bob's user-message for this event now includes Alice's new number.**
If both user-messages are showing trade authorization, number modification continues for Alice and Bob through the numbers specified in their user-messages while their standard modification information is ignored. For example, Alice's message might say "Trading with Anon28849" proceeding with Bob's new number. For extra details or where numbers are too long, users might instead include a hash to some text file uploaded separately.	
If Bob defects, Alice's standard modification information is utilized and her user-message is ignored.	**If Alice defects, Bob's standard modification information is utilized and his user-message is ignored.**

Assigned value: database access in San Francisco

Alice retains database access if defection

— used if defection —

Bob retains board membership if defection

Assigned value: board membership in Moscow

— used if cooperation —

Alice points to a new number only Bob controls

Bob points to a new number only Alice controls.
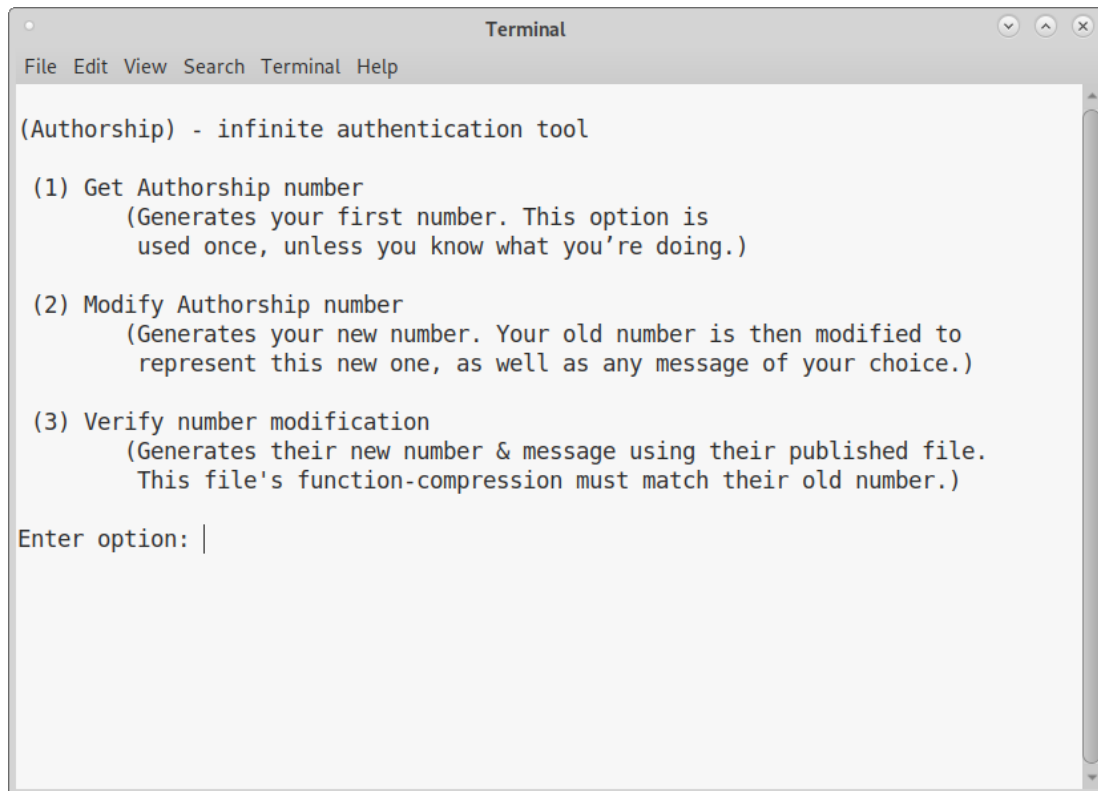
Other operations & information

1. <u>Symbol assignment—missing information is information</u> – There are a total of 126 possible combinations for binary strings of length nine where each string contains (5) ones and (4) zeros. Authorship functions are assigned to "1" if solved and "0" if left unsolved. This is how Authorship extracts meaning from every nine contiguous multi-way functions in a list of 13,500. This list is partitioned the following way. The first 500 groups of nine each produce two digits (00 – 99) of the new <u>Authorship number totaling 1,000 digits.</u> The remaining 1,000 groups of nine each produce one of 95 printable ASCII characters meant to mirror the <u>user's message totaling 1,000 characters.</u> 500 groups of nine + 1000 groups of nine is equal to 13,500 functions in total.

     If you choose to omit a message for some authentication event then instead, the remaining 1,000 groups of nine each produce one and the same value not used in the user's message symbol assignment. Each group of nine functions in the number-partition can express 26 more symbols. And each group of nine functions in the message-partition can express 30 additional symbols since 95 are assigned to characters and one to "null." Later versions may utilize these strings.

2. <u>How to audit the program</u> – *Programming for the Mathematician* is a good resource of mine created for those who like measuring mathematical realities, and wish to weaponize monster numbers loyal only to their summoning wizard. This short book is a minimalist introduction to core C++ and how to work with numbers of arbitrary size and precision. It's composed of three concise chapters and focused on cryptography. Using the bit of knowledge and surprisingly few concepts in this book, you can build anything with the complexity of Authorship itself—and much higher. It allows you to pick up C++ effortlessly so you can read and verify that Authorship.cpp is doing what the documentation file says it's doing. Book license: WTFPLv2+ (see wtfpl.net.)

3. <u>Core values</u> – Encryption strength over convenience, works are single .cpp files written in core C++, no functions, everything in main(), all original & specialized software, widely comprehensible functionality & auditing, and works are never restricted.

4. <u>From digital warfare to the discovery of transparent vaults</u> – Authorship would not have been possible without direct inspiration from my personal heroes who demonstrated beautifully the need for endless cryptographic evidence— Julian Assange and Edward Snowden—this is for you!

```
                              Terminal

 File  Edit  View  Search  Terminal  Help

(Authorship) - infinite authentication tool

 (1) Get Authorship number
        (Generates your first number. This option is
         used once, unless you know what you're doing.)

 (2) Modify Authorship number
        (Generates your new number. Your old number is then modified to
         represent this new one, as well as any message of your choice.)

 (3) Verify number modification
        (Generates their new number & message using their published file.
         This file's function-compression must match their old number.)

Enter option: |
```