stratumn

Proof of Process

A Method for Proving the Integrity of a Process between Partners

September 13, 2016

Abstract

Proof of Process is a scalable protocol that allows multiple partners to trust a common process, or a workflow, by decoupling the proof of data from the secret data in a way that results in a single contextual proof that spans all the steps of a process.

—

In life and the world at large, we see processes everywhere.

A process is any sequence of steps in time. Whenever there is a movement of information, ideas, conversations, goods and products, we have a process.

If we can play back the steps of a process, then we have enabled auditing and traceability. And if every step in a process can demonstrate its veracity to observers, then we have transparency.

Traditionally, when institutions want to share their set of processes with each other, they have to create common bridges to share their data. Those bridges usually consist of APIs, firewalls, and access management.

These bridges leave us two important questions: How can we trust the data? And once the data is trusted, can we reuse the trust?

The prevalence of Software as a Service (SaaS) platforms have resulted in more and more consumers to rely on someone else's processes and systems for their business and data. For such platforms proving that their platform can be trusted to their customers is highly critical.

In this paper we provide a novel solution for sharing processes by introducing a protocol for verifying the veracity of facts in each and every step, enabling a solution with which trust can be packaged, shared and demonstrated easily without the need for sharing the data behind the trust.

Table of Contents

Introduction

Everything is a Process

A process is a sequence of steps where in each step, stakeholders perform a specific action at a specific time in relation to its previous step, or introduce a new element in the sequence.

All steps do not have to have the same set of actors, and the interaction between actors can be asynchronous as long as the interactions can be grouped together into units of steps that follow each other in time.

Additionally, any step can fork into multiple steps without needing to reconcile into a single step. There can be parallel steps if they are performed at the same time but then they will not be in the same thread (timeline) of the process.

As each step captures an instance in time of the state of the system as a whole or in parts, if the process is split in parallel timelines, and since time always moves forward, there can never be circular steps within the same timeline. Within the same timeline, even if in every way two steps match their respective contents, the timestamp in each will be different. For a loop, the timestamp will increase with every iteration.

The examples of processes can be found in almost every human to human and human to machine interactions. Following are some of the examples:

- Trade and Settlement process in the financial space
- The board portals in Software as a Service (SaaS) space
- The Worklist in an enterprise requiring multiple layer of approvals
- Online multi-player games
- Games like Chess and Battleship

In fact if the trust in a system can be established and cryptographically proved to be tamper proof, it could be taken to next level to be shared with any system in distributed fashion. For example if a customer's identity is established with one system and its trust cryptographically proven to be unbreakable, it could be taken to a whole different level where trust could be shared just like data with other systems thus creating "internet of trust."

Battleship: The Process

Consider the board game of Battleship. In this game, there are two players, each having two grids of 10x10 (in our example) cells representing 2D coordinates along with 5 battleships for each player. One grid is for placing the ship, which we will call a ship grid, the other for attacking the other's ship which we will can an target grid.

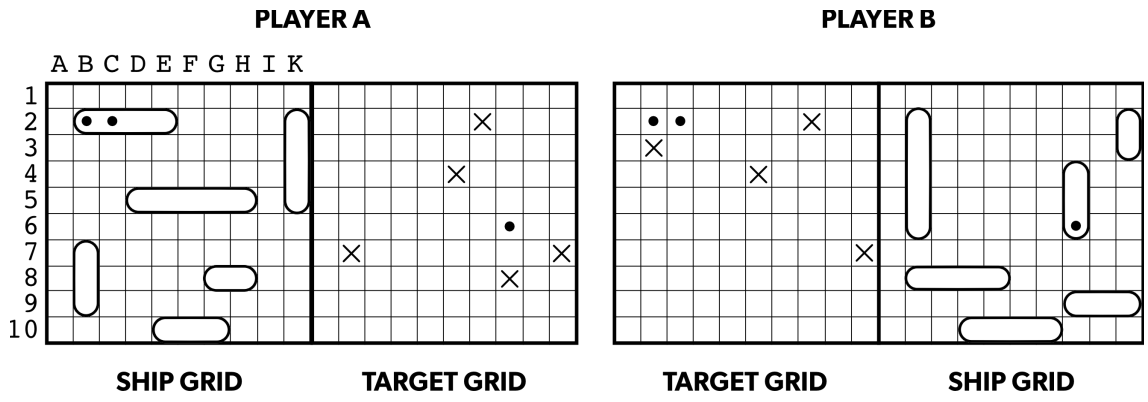

Figure 1. Battleship

An example of how a Battleship game might appear after several turns of play.

The process of the game is as follows:

1. First both players place their ships on their respective ship grid.
2. Each player guesses the location of his opponent's ship by placing pins on his target grid. If it matches one of the opposing player's ships positions then pin is red and the opponent's ship is damaged or sunk, otherwise it's white and the opponent's ship is not affected.
3. Whoever is able to sink all of opponent's ships first wins the game.

Let's say Alice and Bob are playing this game. Each instance of the game being played represents a process with each move representing a step.

In the first step, both Alice and Bob place their ships wherever they like. From then on, the game is played as a sequence of two step couplets: in each couplet, one player is guessing (first step of attacking), followed by the other player responding (the second step of defending) with a match or no-match. In every other couplet, Alice and Bob reverse their roles. If Alice is attacking and Bob is defending currently, it will be followed by Bob attacking and Alice defending, each set of attack-defense forms a couplet.

In every step, there can be either of the two moves: attack or defend. Whoever is making the move right now is the actor responsible for the current step.

Thus every step involved the following parameters:

1. What is being guessed (the coordinates of the other player's ships)
2. Who is guessing
3. When she is guessing
4. Where she is in the game

In each step, each player needs to be able to trust that the other player is not lying. Additionally, each player should be able to prove their move, so they can be rewarded with points for scoring winning moves. At the end of the game, we need to be able to objectively prove the validity of all of the steps for audit and to declare a winner of the game.

Let's say Alice won the game. If Alice wants to reuse her score for gaining reputation, there is no easy way to demonstrate the fact that Alice did win without publishing the record of the entire game with all the moves as attested to by an objective witness.

We can solve this by imprinting the trust from an objective witness in a signed receipt for the data in question, so the data with its receipt can be treated as a fact. Thus, we need a trust system underlying the game where each player can establish the details of their move as an objective fact.

One possible option is to notarize Alice's moves as well as the final result the game (her victory), and then sharing the notarized record as a standalone marker of trust. The result of the game would then be considered a fact.

If we could take the trust between Alice and Bob and package it in some kind of envelope with the following conditions, then we can demonstrate it trust in a decoupled and modular way.

- Anyone can open the envelope verify the fact of her winning the game
- She does not have to reveal her secret moves
- There is minimal involvement of the trust source

Such an option is possible by building what is called a proof system, which we will discuss in detail in the next section. But to clearly grasp the nature of proof systems we must first clearly understand the nature of facts themselves.

Key Concepts

Types of Facts

Facts are statements that represent reality. In order to label a statement as a fact, an honest inquirer must carry out a "fact-checking" experiment to verify if the statement indeed represents reality accurately. If so, then the statement is qualified to be labeled as a fact. These are known as *a posteriori* facts. Scientific and empirical facts with experimentation as its basis of fact-checking are also examples of *a posteriori* facts.

However, there is another breed of fact: *a priori*.

A priori facts represent reality which can not be experimentally tested, such as the mathematical fact of "2 + 2 = 4." Indeed, a "fact-checking" procedure would represent a deductive logic following the rules of decimal computation. Another example of an *a priori* facts would be "one coin goes in a transaction, one coin comes out."

Now in practice, apart from the above two breeds of facts, *a posteriori* and *a priori*, we also need to establish facts in places where there can not be any sort of fact-checking. In the inner world of feelings we have to depend on a trust source, that being the person telling us how she feels or an source of authority attesting the claim. Another example is the fact of who made which move and who won at the end of a game of Battleship. It is both computationally impossible as well as experimentally impossible to fact-check if both players conspire together to make up false moves in a logically consistent way. We refer to these kinds of facts as subjective facts.

Facts in Digital Systems

With *a priori* facts, digital systems can compute a fact-checking algorithm to establish if the statement (aka string) is indeed a fact. For example, an algorithm can check if the number of coins going in a transaction equals the number coming out. Any kind of computation to verify an *a priori* fact will be deterministic in nature. However, for *a posteriori* facts and for subjective facts, we have to depend on some trust source as the basis of "fact-checking."

In the context of digital systems, facts are either computationally verifiable datasets (*a priori* facts) or trusted datasets (*a posteriori* and subjective facts).

Once a fact is established either by a fact-checking algorithm or through a trust source, there is a need to demonstrate its veracity if it is challenged later. We can either repeat the fact-checking experiment, or we need to be able to trace back all of the steps in the history of events leading to the original event of its attestation by the trust source as a fact.

Thus, we can say that first there is the question of how to establish a dataset as a fact, either by fact-checking or via a trust source, then comes the question of how to prove the fact to be so whenever its veracity is challenged. Thus, proof comes after the fact.

For *a posteriori* facts in digital systems, being models of reality, we can not run an actual experiment to verify their factuality. So we need a trust source to attest a dataset as a fact, in the same manner as subjective facts. Once attested, a proof system can then be used to demonstrate the trust that was responsible for the attestation of the fact. The trust source can be a centralized authority, such as a government authority, or a decentralized network of consensus such as the authority of ancient traditions or that of a blockchain timestamp server.

The proof of process does not establish trust but provides for a way to demonstrate the trust that already established the fact to be so in the first place. A proof exists only in relation to facts getting challenged, however, trust lives on independently and, ideally, objectively.

In this paper, we do not seek to establish trust, as it can be done either by fact-checking (by computing an algorithm for *a priori* facts such as checking input coins vs output coins in the bitcoin blockchain) or by receiving a signed attestation from a trust source.

Instead we focus on building a proof system which can only be used after the establishment of the fact to demonstrate the trust.

Proof Systems

A proof system is what enables a first party (called a 'Prover') to exchange messages with a second party ('Verifier') to convince the Verifier that the subject of the proof to be true within the context of their mutually agreed upon source of trust.

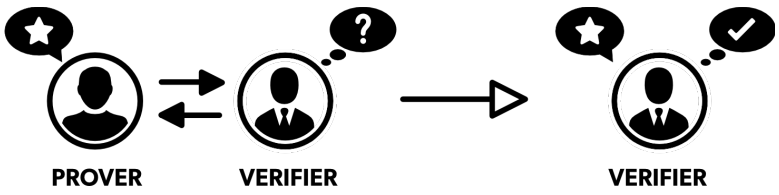

Figure 2. Proof System

There can be two kinds of proof systems.

For *a posteriori* and subjective facts a proof system can be made to establish a protocol for capturing enough information from a process to build a proof that can demonstrate the trust that established the process in the first place.

For *a priori* facts: being computationally verifiable, by enabling code execution for validating each step of a process where the code execution is deterministic in nature. Thus, the validation logic acts as the proof system.

For the purpose of this paper, we only focus on the former: proof system for *a posteriori* and subjective facts. And we are not checking if the facts were established in an honest way. However, for established facts, we are enabling a proof system to demonstrate who established what, when, where, and how. These facts are in the context of a process; thus, the proof system covers all of the steps of process.

Proof of Process is a scalable protocol that allows multiple partners to trust a common process, or a workflow, by decoupling the proof of data from the data in a way that results in a single contextual proof that spans all the steps of a process.

In fact there can be a separate thread, one for the actual process with factual steps, another for the proof system corresponding to the actual process. Thus, the trust source that attests datasets as facts through fact-checking in turn builds a separate thread for enabling proofs to verify the veracity of the facts.

By separating the proof of data from the data itself, the proof of process can exist in parallel to the actual process. The proof of process can contain minimal information, and no sensitive data from the actual process but can be used to verify the veracity of the facts of the actual process.

Design Rationale

There is the actual process (the game being played) where Alice and Bob make their secret moves, and this remains secret. Then there is the proof of that process which is derived by extracting just enough information from each step of the actual process without including the actual moves in order to build proofs to verify each step objectively.

It is necessary that we address key security concerns as we build the proof, such as:

1. **Data Integrity**
   Data Integrity refers to the ability to prove that the content in each step corresponds with its intended step, that is there has been no corruption or tampering of data. So if the data integrity of a step is challenged, the stakeholders can prove that they are referring to the correct information content. This is a Proof to demonstrate the **What** of a step.

2. **Non-repudiation of Source and Destination**
   This refers to the ability to prove the identities responsible for a particular step in such a way that the source or the destination of the information content of a step cannot be repudiated. This is a Proof to demonstrate the **Who** behind a step

3. **Proof of Anteriority**
   This is the ability to prove when the step was performed. This is a Proof to demonstrate the **When** of a step.

4. **Contextual Proof**
   This is the ability to unquestionably demonstrate where a step belongs in the sequence of steps of a process, i.e., prove the correct sequencing/ordering of steps. This is a Proof to demonstrate the **Where** of a step.

A single proof for each step of the process is possible if we can consolidate the individual proofs for each of the four information security concerns into a single proof per step. Once we have a single proof per step, if we can consolidate the single proof per step for all steps into a single proof for the entire process, then that would be the Proof of Process.

Enabling Data Integrity

In information security systems, data integrity implies maintaining and assuring the accuracy and completeness of data over its entire life-cycle. This means that the stakeholders should be able to prove that the data has not been modified in an unauthorized or undetected manner. The most common tool to accomplish this is the cryptographic digest.

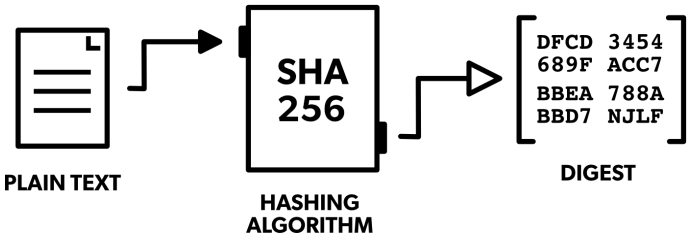

Figure 3. Cryptographic Digests

A cryptographic digest process takes a plain text and generates a string unique for that input text, thus acting as a sort of a fingerprint. However, the generated string looks nothing like the original text. The generated string is known as the cryptographic digest string or simply, the digest.

The digest can be used to verify if the plain text has been modified. This is known as an Integrity Check. However, it cannot be used to recover the original text. Being a fingerprint, it also assists in locating duplicate texts.

Let's take the example of the game Battleship. Let’s say the first move by Alice is "D10."

To create a digest of Alice's first move, we will pass the string "D10" through a cryptographic hashing algorithm such as SHA-256 to generate a digest, then store the digest in that step.[1] Alice will store her actual move separately as it does not need to be part of the proof of process; we only need to store the document digest. She gets to keep her moves secret.

As the digest will be unique for that specific move, we can use the digest to prove that the move remains intact or that this indeed is the correct move we are looking for. Whenever challenged as to the veracity of the digest in a step, Alice can just take her secret move to generate the digest and demonstrate that there is a match between the generated digest with the digest stored in that step. In this way, Alice can demonstrate the proof of What in each step.

Enabling Non-Repudiation of Source and Destination

Non-repudiation implies that the stakeholders of the information content of each and every step should not be able to deny their involvement with the steps representing their data through the digests. The tool we will use for this is Digital Signatures.

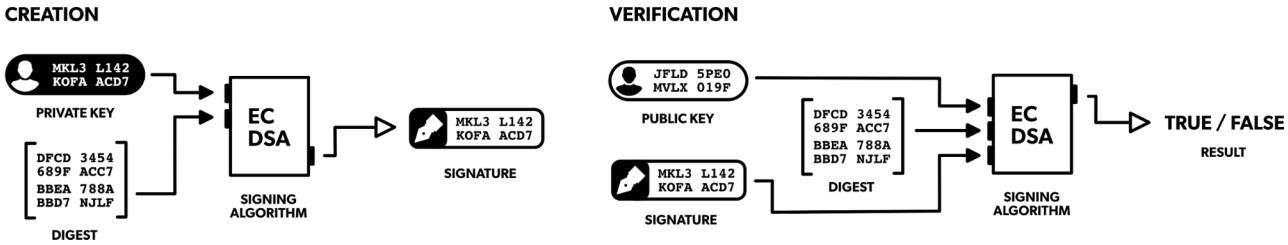

Figure 4. Digital Signatures

First you need a split key pair: a private key with its corresponding public key. Then you pass the digest with the private key through a signing algorithm to create a signature.

The public key can then be used to verify if the signature matches the digest. However, the public key cannot be used to "sign" the digest; only the private key can sign. The public key cannot be used to recover the private key. The signature does not contain the input text nor can it be used to recover the input text.

To verify a signature you need the digest along with the signature and the public key corresponding to the private key used to sign. The private key is not required to verify and is never shared. Only the signer has access to her private key.

Both Alice and Bob need to be responsible to their respective steps in such a way they they can not repudiate their involvement if challenged. The record of their identities would be maintained by having the stakeholders digitally sign the digest of their move and then storing the signatures and public keys along with the digest in the step. The private keys will not be stored in the steps; each player hold hers

[1]To prevent brute force attacks, especially on a small set of moves such as battleship, a nonce should be included in the move before cryptographic hashing

separately and securely. Anyone who has access to the proof can use the public key verification to ascertain whether or not Alice or Bob can be held responsible for a step.

In this way we enable identity management and ownership in each and every step for the proof of a process to demonstrate the Who behind each and every step.

Enabling Proof of Anteriority

Proof of anteriority implies the ability to prove when a piece of information was certified or signed. To enable this, it is necessary to be able to prove when the signature was done, as keys can expire or the document behind the digest step can expire. Thus, the validity of a piece of information is dependent on time. So we have to introduce the parameter of time in our proof in a secure way.

To reflect the linear arrow of real time in our system, we must ensure that once a step is performed, it cannot be reverted back. Only new steps can be added and older steps are never removed. This makes each step immutable.

Within each immutable step, if the time the step was performed is included in its digital signature, it must reflect its real time, as the recorded time cannot be changed at a later date because it would invalidate the signature.

We can obtain the time from a trusted time source who attests the time before it is finally recorded in the step to make the time neutral to all the actors of the process. This is referred to as trusted time-stamping. (See following page for more information on trusted time-stamping.)

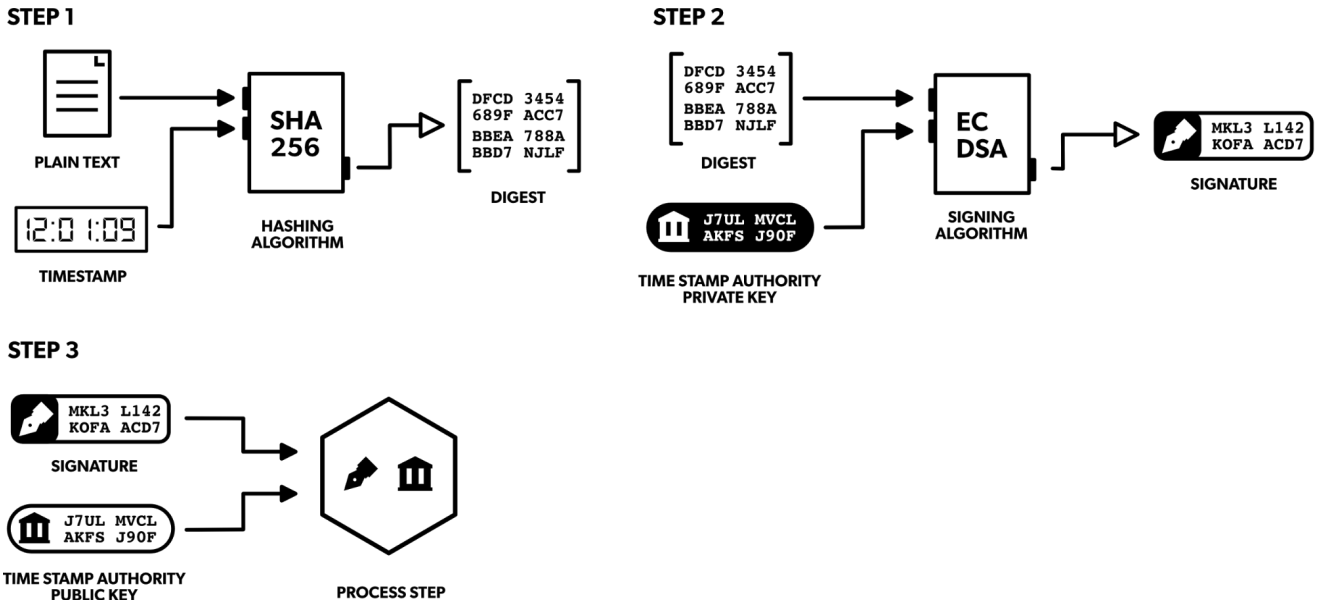

Figure 5. Trusted Time-stamping

For each step, before signing the digest, the proof system will include the current time inside the digest. Thus, the stakeholder has to sign the digest with the time included just like a signature on a paper document contains the date next to it.

The time has to be recorded by the trust source, so we include the signature of the trust source too. Once both parties sign, we would then store the time along with the public keys, signature and the digest in the step, all of which can be used to prove the veracity of the signature if challenged.

In the absence of a trust source, the timestamp can generated locally but has to be signed by the majority of parties involved in the process to get a consensus on a common time.

Enabling Contextual Proof

For each step, in addition to the digest, the digital signature, the timestamp, and the public key, there is also the context surrounding that step.

The context for a step is what makes the step part of the process that it belongs to, and that which places that step in relation to its previous and following steps. Thus, it is the envelope around each step where we store the data that represents which process the step belongs to, and where in the process does it belong.

In order to ensure the sequencing of steps has not not be tampered or changed, we enable Contextual Proofs. It is the proof to demonstrate the Where for each step.

This is achieved in two steps:

1. Creating a single proof string for every step with its context included. We call this **linkhash**.

2. Including this linkhash of the previous step in the context of the current step, thus creating a chain of contexts, or Context Chain.

LinkHash

To get the single string that represents the entire proof of a step with its context, we pass the entire contents of a step through a hashing algorithm to generate a digest that outputs a single string that is unique for that step and its context. That digest is the linkhash.

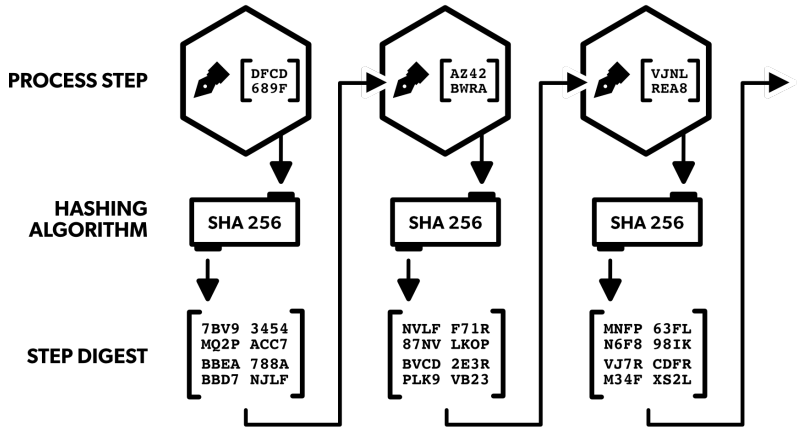

Figure 6. LinkHash

The digest of all of the contents of each step with its context captures the entire proof of that step in a single standalone shareable string as the contents of the step are essentially elements representing the proof. This single proof string for a step is known as a linkhash.

Context Chain

With each step we include a “link” to its previous step by including a single string that represents the entire proof content of this previous step, the single string being the linkhash from its previous step.

For the first step in the process, linkhash of its previous would be empty as there is no previous. For the second step, the linkhash of the previous step would be the digest of the contents of its previous step which is to be included in the context of this step. Thus, we are creating a chain where the context of every step is bound to its previous by including the linkhash of its previous step. This enables Contextual Proof of each step.

Thus the contents of each step are:

1. Digest of the Information Content of that step to enable Data Integrity
2. Trusted Timestamps to enable Proof of Anteriority

3. Digital Signature with the Public Key to enable Non-Repudiation
4. Linkhash of the previous step in its context in order to enable Contextual Proofs

By including the linkhash of the previous step in the context of the current one, the context of each step includes the digest of: the digest, signature, timestamp, public key, context of the previous step - context within context going all the way back to the first step.

Cumulative Proof

By including the proof of its previous step, every step in effect contains the proof of all its previous steps, forming a Cumulative Proof. Thus, we have a Context Chain of Cumulative Proofs to ensure the Contextual Proof.

With Cumulative Proofs, when we share the proof of a specific step, we have effectively shared the proof of all its previous steps. Alice can just carry the final linkhash which will contain the proof of all the steps of the process. The final linkhash can act as the definitive proof instead of each of the proofs in all of the steps.

Cumulative Proof creates a chain of proofs which makes each proof tamper-proof as to change one, you would have to go all the way back to the first step to change the proofs in each one of them, which would in turn invalidate the entire process.

The benefits of Context Chain of Cumulative Proofs are:

1. Sharing the proof of a step in turn shares all its previous proofs
2. The impossibility of altering a single proof without invalidating all of its previous proofs
3. Directional Proofs: Proofs have a direction and strict ordering from the first to the most recent proof as they are computed
4. Path of Proof: Not only can you demonstrate the proof of a step but you can also prove the path it took to get to that step

As we can represent an entire proof of process by the linkhash of its final step, we can include an entire proof of process inside one of the steps of another proof of process, thus creating nested proof of processes. Then the linkhash of the final step in the outer proof of process will include the proofs of all its steps including the proofs of the steps of the inner proof of process.

In this way, we can connect multiple proof of processes. For example, the proof that Alice won can be connected to the proof of her winning another game to build her reputation profile.

Building the Proof of Process

Battleship: Proof and Process

Step 1: Initial State

At the beginning of the game, we store the digest of the position of the ships for both Alice and Bob with both players' signatures on it. This is the first step of the proof of the process.

The actual data behind the digest, which are the positions of the ships, can be stored by each player separately and kept in an offline space.

Step 2: Alice Attacks

Alice makes her first move: she calls out "D10" to attack Bob. We record the digest of the string "D10" along with the time when Alice called it, and then apply Alice's signature on the digest with the timestamp on it. This information becomes the second step of the process and contains:

- The digest of the string "D10"
- Alice's signature
- Trusted timestamp
- Alice's public key for signing
- The linkhash of the first step in the context of the current step

The linkhash of the second step, containing all of the above parameters stored in it, acts as the unique fingerprint representing its proof of existence without having to store the secret data (ship positions) behind it.

Step 3: Bob Defends

Bob responds with a match and puts a red dot on top of the ship at D10 of his ship grid indicating that Alice scored a red point on her target grid. We record the digest of the string "match" along with the time when Bob responded, and then apply Bob's signature on the digest with the timestamp on it. This is the third step where we save:

- The digest of the string "match"
- Bob's signature on the digest with the time on it
- Trusted timestamp
- Bob's public key for signing

- The linkhash of the second step in the context of the current step

Going Forward

The second and the third step form the first couplet of this instance of proof of process. In the following couplet, they will reverse their roles: Bob will attack and then Alice will respond, forming the fourth and the fifth steps respectively. This will go on till one of the players has sunk the other's ship, making the survivor the winner.

In the final couplet, when Alice makes her winning move, Bob responds with a match. We will store the details for that couplet, and in addition, we will store an extra step as the final step containing:

- The digest of the string "endgame"
- Both Alice and Bob's signature on the digest with the time on it
- Trusted timestamp
- Both the public keys for signing
- The linkhash of the previous step, where Bob replied with a "match"

The linkhash of each and every step acts as the unique fingerprint representing its proof of existence.

If the context chain is built by a computing platform common to both the players in a way that every time the players calls out their moves, it can compute the proof. Computing the proof for each step involves: hashing the move, getting the players to sign their respective moves, adding trusted timestamping, and building the context chain with the proof of every new move linked to its previous move.

The computing platform can publish the linkhash of the final step of "endgame" which acts a proof for the entire game, while not sharing the rest of the hashes with other players. Each instance of the entire context chain can be uniquely referred to by the final linkhash so the players can just share the final linkhash with other players as a proof of the game's history.

Now that we have the proof of process (demonstrating the fact of winning) decoupled from the actual process (the private record of Alice's moves), we have to publish the context chain of cumulative proofs in a way that the proof can be publicly verifiable without revealing the secret data behind the proofs so everyone believes Alice's track record without being able to use her secret moves.

Facilitating Zero-Knowledge Proof of Process

To enable a reputation based system for a community of gamers, we need to enable a shareable record of winnings and scores for each gamer. We can start by using the proof of process to verify the honesty of each game being played. However, to make sure that any set of two players does not make up fake games for reputation, or that gamers don't make up fake players to win against, we need to introduce referees sitting in each game, who could be other gamers not playing at that moment. If

there is a referee attesting each move of a game and thus each step of a proof of process, then we can use a "two out of three signings" to validate the honesty of a step. The three in this case would be: Alice, Bob and the referee. Alice can now present the proof of process to anyone else for verification, and they can just check the "two out of three signings" for verification.

Thus, for a peer to peer system of interaction, like that of Alice and Bob, we need a third person to act as an arbitrator. However, for a system where the parties involved are hierarchical, we do not always need a third party. Consider the case of Alice going to a bank to open a new bank account and thus going through a Know-Your-Customer (KYC) process. Because the bank also acts as an arbitrator as it is also the Trust Source. So for generating a shareable proof of process in a hierarchical system such as the KYC process, only two parties are enough.

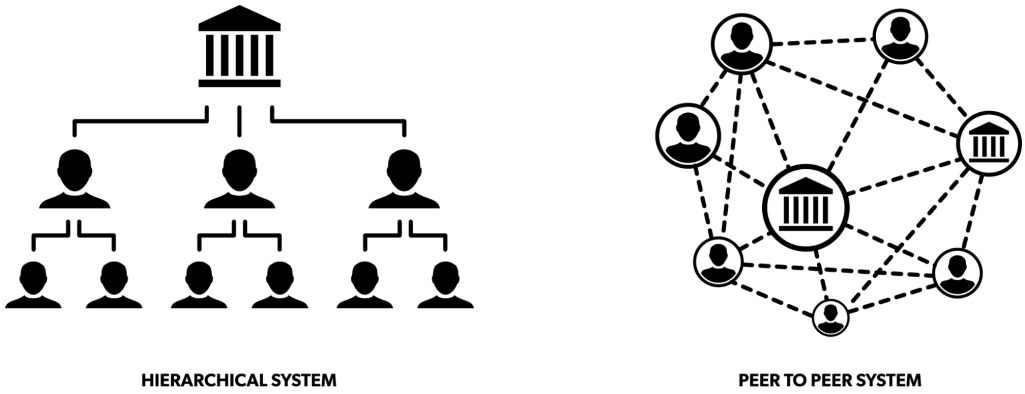

Figure 7. Hierarchical and Peer-to-Peer Systems

Most organizations function with hierarchical structures where multiple parties are grouped under the same head, whereas in peer-to-peer systems, information is shared sideways and directly between its members making the exchange often faster. However, in peer-to-peer systems trust becomes difficult to implement without a trusted head like that of a hierarchical system. Proof-of-process draws upon the strengths of each system to enhance the benefits to the individuals operating in both systems while also enabling interaction between both system.

During the building of the proof of process, to prove who she is along with her financial history, Alice has to share her private documents with the bank, such as her Birth Certificate. Once a document gets validated by the bank, the bank attests the step for it with its signature. Thus each step performed by Alice will have the bank's signature attesting to its authenticity.

Having completed the KYC process with this bank, when Alice goes to another bank to open another account, this new bank can just use the proof of process that Alice received from the first bank as the second bank will trust the first bank because the banks, being peers, are in the same level of trust for each other, while Alice is under both. Additionally, Alice can sign a new message for the new bank with her private key and prove she is indeed the actor in question in the proof of process build by the first bank. The new bank can then check if the new signature matches that in the proof of process without needing to access her private key for signing. If the new bank challenged Alice on the integrity of the

content of one of her documents, she can just generate a fresh digest from her private document and present it to the bank for verification.

So if every time a new bank challenge's Alice's honesty, she can just provide the proof of process with her attesting signatures, along with those of the bank's on the digest of her private documents, without having to share the private documents in plaintext (plaintext that she had to share with the first bank). Thus the private documents remain private for every new bank. Even for the first bank, if they have to share Alice's Risk Score, as generated in the KYC process, with another department in the same bank, they can do so without having to share Alice's private documents. This is because they can verify the honesty of her score with a complete audit trail which is the sequence of steps of the proof of process.

With this KYC example, Alice is able to proof her honesty in the Proof of Process from the first bank to the new bank without having to reveal her secret data (her private documents), and with the Battleship example of "two out of three signings," she is able to prove she won (for gaining reputation) without having to reveal her secret moves. These are classic examples of Zero-Knowledge Proof of Processes in a hierarchical system and a peer to peer system respectively.

Figure 8. Zero-Knowledge Proof

For each and every step of a proof of process, if we do not store any secret behind the proof but only the elements to enable the proof (those being the digest, signature, public key and the trusted timestamp) while still ensuring that the proof can be demonstrated to any verifier, then we have a zero-knowledge proof of process.

Proof of Process does not necessitate, but it provides a framework for enabling Zero Knowledge Proof.

With a zero-knowledge proof, even after publicly verifying the proof, the verifier cannot duplicate the proof as the verifier does not have access to the secret (Alice's actual moves or her private documents, and her private key to sign) behind the proof.

As the zero-knowledge proof of process (ZK PoP) produces a complete audit trail by default where facts in each step can be proved, if challenged, without having to reveal the secret data behind the proofs, we have enabled traceability, auditing and transparency in the system.

And with that, peer-to-peer systems can interact and exchange data with hierarchical systems in a seamless way within the context of trustability where each system can have its own source of trust, thus enabling a modular and decoupled trust network.

With a zero-knowledge proof of process, we can publish the context chain of cumulative proofs in a publicly verifiable way as no secret needs to be shared for the proof to be demonstrated; whatever is publicly accessible is enough to demonstrate the proof to any verifier.

Notarization

To notarize, a typical solution has the need of an objective witness in the form of a trusted third party to act as one the signers. However, if we distribute the storage of the context chain across multiple parties, then anyone can verify the veracity of each and every step because the proofs are verifiable without the need of the secret, making the proofs publicly verifiable.

We can store the entire context chain of cumulative proofs in a public store without losing any security benefits:

- As it is a cumulative proof, it is increasingly difficult to tamper the proofs as we move forward in the process.
- The context chains enable a complete audit trail, so we can go back to an earlier version in case of any discrepancy.
- Neither the private key for signing nor the plaintext behind the digest needs to be made public for verifying the proof.
- The context chain demonstrates the ability to verify the proof without the ability to create and thus duplicate the proofs.

Therefore we can have a network of verifiers where truth can be decided based on the consensus amongst them about the veracity of the publicly verifiable proofs instead of a single objective witness. This is known as Consensus Mechanism.

As long as the majority in a consensus network agree with the trust source, it can be used to trust the proof without having to go through proving it every time it is challenged. Additionally, with consensus, we do not need to trust a single source of verification of the proofs; there can be multiple verifiers spread across a network, adding to the trust in the proof of process.

Instead of having to trust a single source of truth like in typical notarization, we can maximize the trust by consensually verifying the proof of process by multiple parties as long as the stakeholders for each step sign their involvements because anyone in the consensus network can pose a challenge to verify the proofs without needing to know the secret data behind the proofs.

The separation of the proof of data from the actual data, coupled with distributed notarization (by consensus), paves the way to enable trust in heterogeneous networks by minimizing the need for expensive bridges such as that of APIs, firewalls, and access management.

As the proof in each and every step always already includes the proofs of all of its previous steps, by notarizing the final step we have effectively notarized the entire proof of process. Alice can use the linkhash of the final step to gain reputation based on her track record (to enable trusted communication between other Battleship players) without revealing any secret moves as she possesses the ability to demonstrate the trust whenever challenged. This lets us share minimal information with the maximum benefits of information security.

Conclusion

By separating the proof of process from the actual process, managing changes in a process becomes easier. It helps to manage trust in a focused way without getting overloaded with the data behind the trust. And we can utilize nested proof of processes to simplify for trust systems with dependencies on other trust systems.

Another common information security concern for establishing trust is Confidentiality, which implies making sure that no one unauthorized can have access to the data. In our example, this is a non-issue as we are not storing the actual moves, just the digest of the moves.

However, if Alice wants to share her secret moves behind the proof, she can then encrypt the list of her moves using the public encryption key of whoever wants to read, and then share the encrypted data. In this way, she still maintains the separation between the proof of her winning from the sharing of her secret moves.

So to recount the steps for generating the proof of process from the actual process:

- Extract the trust by deriving proofs with respect to the key information security concerns for each step of the actual process
- Enable Data Integrity through cryptographic hashing
- Enable Non-Repudiation of Source and Destination with Digital Signatures
- Enable Proof of Anteriority with Trusted Timestamps
- Enable Contextual Proof with Context Chains
- Create a single proof per step: As each of the sub-steps in the above step builds on its previous, we are naturally consolidating into a single proof as we go along
- Consolidate a single proof for the entire process with Cumulative Proofs
- Publish the Proof of Process in a distributed manner, using a consensus mechanism as the source of truth

In today's world of technology, there are no islands of data but continuous movement of information, ideas, conversations, goods and product underlying which, there needs to be a foundation of trust that enables the honest exchange of items. If we can manage trust in a decoupled and modular way in every exchange through a protocol to verify the veracity of facts in any exchange, we have the next paradigm for markets.

Proof of Process was initially developed by the Stratumn team in efforts with their customers and partners. Its authors are Anuj Das Gupta, Richard Caetano, Akbar Ali Ansar and Stephan Florquin with editing and illustration by Gordon Cieplak.

A framework to enable the proof of process protocol can be found at Chainscript.io