Proof of Process

A FOUNDATION FOR NETWORKS OF TRUST

The White Paper

–

Anuj Das Gupta, Richard Caetano, Akbar Ali Ansari, Stephan Florquin & Gordon Cieplak

Abstract

PROOF OF PROCESS IS A PROTOCOL that allows participants to trust a common process by decoupling the proof of data from the actual source data in a way that yields a single proof that represents all steps of the process.

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

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

More and more organizations and individuals rely on these aging bridges every day to handle data that governs the processes in both their personal lives and business operations.

In this paper we provide a novel solution for sharing processes by introducing a protocol for verifying the key factual elements of every step in a process, enabling a solution through which trust can be easily packaged, shared and demonstrated in a way that enables traceability, compliance, privacy, and accountability.

Introduction

Everything is a process.

A process is a sequence of steps in which actors perform specific actions at specific times relative to prior steps or introduce new factual elements and process actors in the sequence.

All steps need not include the same set of actors, and the interaction between actors can be asynchronous as long as the interactions can be grouped together into sequential steps.

Additionally, any step can fork into multiple branches without any need to reconcile them into a single branch. There can be parallel steps in different branches if they are performed at the same time. As time always moves one direction, there can never be circular steps within the same branch.

Examples of processes can be found in almost every human to human and human to machine interaction. Just a handful of examples from this ocean include:

1. Trade and settlement processes
2. Software as a Service (SaaS)
3. Online multi-player games
4. Board games like chess and Battleship

Battleship: The Process

Consider the board game Battleship. In this game, there are two players, who each have two 10x10 grids of cells representing 2D coordinates, along with five ships. One places his own ships on his primary grid, and uses a tracking grid to record the results of his attacks on the opposing player's ships.

The process of the game is as follows:

1. First, each player places his ships on his primary grid.
2. Each player then takes turns guessing the location of his opponent's ships, and the opponent announces whether or not the location is occupied by a ship.
3. If it matches one of the opposing player's ships' positions, then the opponent's ship is "damaged", which is indicated through red pins on the respective tracking and ship grids.
4. If the guessed position does not match a ship, then a white pin is placed to denote a miss.
5. Whoever is able to "sink" all of his opponent's ships by guessing all the correct ship positions first wins the game.

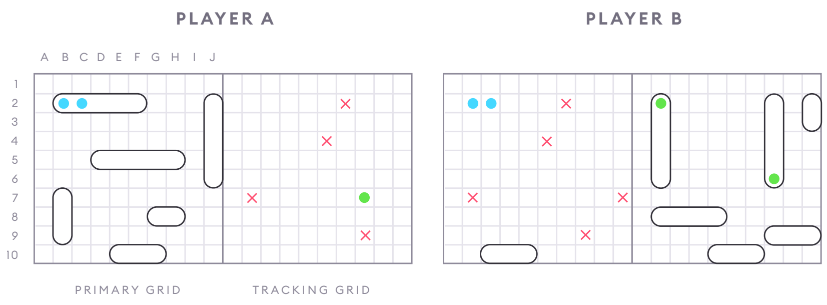

Fig. 1: Battleship. An example of how a Battleship game might appear after several turns of play.

Let's say Alice and Bob are playing this game. Each Battleship match represents a process, while each move represents a step in that process.

In the first step, both Alice and Bob place their ships wherever they like. From then on, Alice and Bob alternate turns of play until a winner is established.

Thus every step includes the following parameters:

- *What* is being guessed (ship coordinates)
- *Who* is guessing
- *When* the guess is made
- *Where* this guess lies in the total sequence of guesses

IMPLICIT CONDITIONS

In each step, each player needs to be able to trust that the other player is not lying. Furthermore, each player should be able to prove their move so that they can be rewarded with points for scoring winning moves. And before we can declare the winner of a game in a meaningful way, we need to be able to objectively prove the validity of all the game's steps.

Let's say Alice wins the game. If Alice wants to use her score to establish reputation, there is no easy way to demonstrate her victory without publishing the record of the entire game along with attestations for each move by an objective witness.

We can resolve this situation by imprinting the trust from an objective witness into a signed receipt for the data in question, so that the data with its receipt can be treated as a fact: digital notarization. Therefore, we need a system underlying the game through which each player can establish their moves as objective fact.

One possible option is to notarize Alice's moves as well as the game's outcome (her victory) and to then share that notarized record as a standalone marker of trust. The game's outcome could then be considered a fact.

Furthermore, we can demonstrate the veracity of the game's outcome in a decoupled and modular way if we can share the trust established between Alice and Bob in their game-play through a package with the following conditions:

- Anyone can open the package to verify the outcome of the game
- No player must reveal their guesses or ship positions
- There is minimal involvement of the source of trust

Such an option is possible through the construction of a proof system, which we will discuss in the next section.

Key Concepts

TYPES OF FACTS

Before we begin to construct a decoupled, modular proof system, let us examine the nature of the facts with which we will be working.

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 the experiment yields a positive result, then the statement is qualified to be labelled as a fact. These are known as *a posteriori* facts. Scientific and empirical facts with experimentation as their basis of fact-checking are also examples of *a posteriori* facts.

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

A priori facts represent reality that cannot 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.

Apart from *a posteriori* and *a priori* facts, we also need to establish facts in places and situations in which fact-checking cannot take place. In the inner world of feelings we have to depend on a source of trust, e.g. the person telling us how she feels, or a source of authority attesting a claim.

Another example includes the facts of who made which move and who won at the end of a game of Battleship. It is both computationally and experimentally impossible to fact-check if both players have conspired 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

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

With *a priori* facts, digital systems can compute a fact-checking algorithm to establish if the statement is indeed a fact. For example, an algorithm can check if the number of coins going into 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* and subjective facts, we must depend on some source of trust in place of experimental or computational fact-checking.

A posteriori facts, in digital systems, being models of reality, cannot be fact checked through experimentation. So, we need a source of trust to attest a dataset as a fact, in the same manner as subjective facts. Once a fact has been attested, a proof system can then be used to demonstrate the trust that was the origin of the attestation of the fact. The trust source could be a centralized authority, such as a governing body, or a decentralized network of consensus such as the authority of ancient traditions or that of a blockchain timestamp server.

In this paper, we do not seek to establish trust, which, as we have seen, can be done either through fact-checking, signed attestation from a source of trust, or decentralized consensus. We will instead focus on building a proof system which can be used after the establishment of facts to demonstrate the trust at their origin.

PROOF SYSTEMS

A proof system enables one party, called a prover, to exchange messages with another party, a verifier, in order to convince the verifier that the subject of the proof is true — within the context of their mutually agreed upon source of trust.

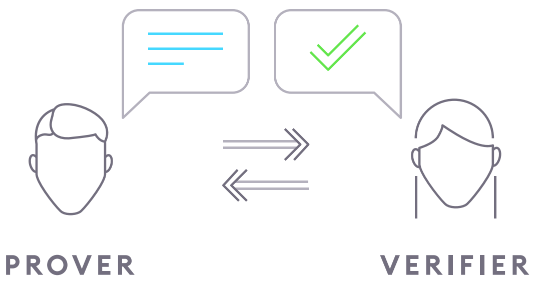

Fig. 2: A Proof System.

There can be two kinds of proof systems.

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

A priori facts, being computationally verifiable, can be validated through the execution of code. Thus, the validation logic acts as the proof system.

For the purpose of this paper, we will only focus on the former: proof systems for *a posteriori* and subjective facts. And we are not checking if the facts were established in an honest way.

That said, we enable a proof system that can demonstrate the key factual elements of each step in a process with established facts in a decoupled and modular way.

Design Rationale

In the example just described, there is the actual process (the game being played) where Alice and Bob each make their own guesses.

Then there is the proof of that process. This proof is derived by extracting just enough information from each step of the actual process, without including the actual guesses, in order to build cryptographic proofs that can be used to verify the integrity of each step by an objective party.

To build these proofs it is necessary to address key information security concerns by establishing key factual elements within the proofs.

KEY INFORMATION SECURITY CONCERNS AND FACTUAL ELEMENTS

1. Data Integrity
The information content of a step in the process must correspond exactly to its intended step in the recorded process in way that proves there has not been any corruption or data tampering. This demands a proof to demonstrate the factual element of what is in a step.

2. Actor Non-Repudiation
The actors responsible for a step in the process must be recorded in such a way that the source of the information content of a step cannot be repudiated. This demands a proof of the factual element of who acted in a step.

3. Proof of Anteriority
The time at which a step in a process occurs must be provable. This demands a proof to demonstrate the factual element of when a step happened.

4. Proof of Context
Where the step belongs relative to the context of all other steps in a process must be unquestionably demonstrable. This demands a proof to demonstrate the factual element of where a step happened.

It is possible to create a single proof for each step of the process by consolidating the individual proofs for each of the four information security concerns into a single proof for each step.

Once we have a single proof for each step, we can then consolidate these again into a single proof for the entire process.

Data Integrity

In information security systems, data integrity implies the maintenance and assurance of the accuracy and completeness of data over the system's life cycle. This means that the users should be able to prove that data has not been modified in an unauthorized or undetected manner. The most common tool to accomplish this is the cryptographic digest.

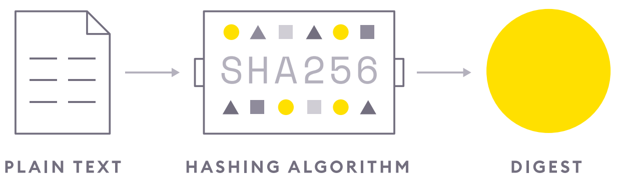

Fig. 3: Cryptographic Digests. A cryptographic digest process takes plain text and generates a string unique to that input text, which acts as a digital fingerprint.

Let's return to our Battleship game. 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 and then store the digest in the step1 of the process graph. Alice can record her actual guess 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 guesses secret — or at least between her and her opponent1.

As the digest will be unique for that specific guess, we can then use it to prove that the guess is legitimate, or that this indeed is the correct guess that we are looking for. If the veracity of a digest in a step were challenged, Alice could use her secret guess to re-generate the digest, thus demonstrating that there is a match between the generated digest and the digest within the step recorded in the process graph. In this way, Alice can demonstrate proof of the factual element of what has happened.

Actor Non-Repudiation

Non-repudiation implies that the sources of the information content of each step in a process should not be able to deny their involvement with the steps that represent their data through the digests. The tool we will use to address this is the digital signature.

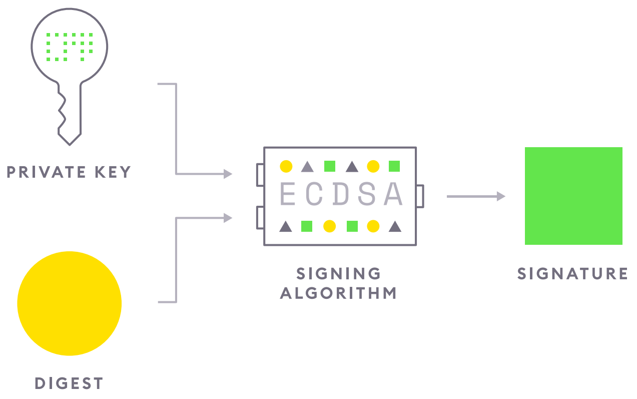

Fig. 4: Digital Signature Creation. First, one creates a split key pair – a private key with its corresponding public key – using a PKI system. Then one uses a signing algorithm to generate a signature from the private key and cryptographic digest.

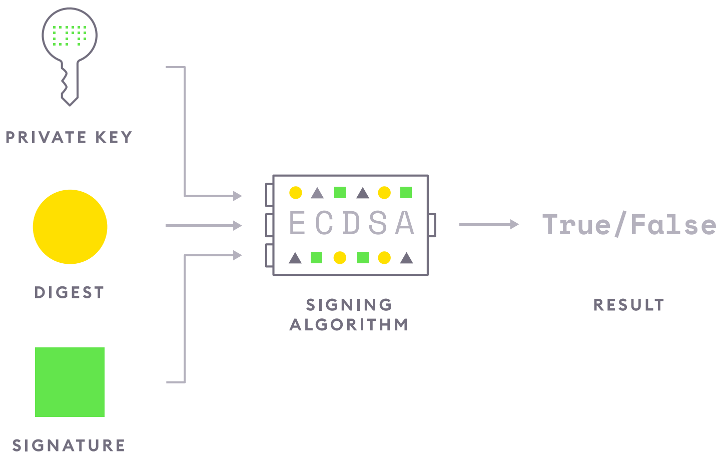

Fig. 5: Digital Signature Validation. The corresponding public key can then be used to verify if the signature matches the digest. However, the public key cannot be used to “sign” the digest, nor can it be used to recover the private key.

Non-repudiation implies that the sources of the information content of each step in a process should not be able to deny their involvement with the steps that represent their data through the digests. The tool we will use to address this is the digital signature.

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 occurred, as keys or the document behind the digest step could expire. Thus, the validity of a piece of information is dependent on time. So, we must introduce the parameter of time into 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 to its previous state that only new steps can be added and older steps are never removed. Steps must be immutable.

If the time the step occurred is included through a digital signature, it must reflect the real time. The recorded time must not be able to be changed at a later date — this should invalidate the signature of the step.

We can obtain the time from a trusted time source that uses digital signatures to attest the time before it is finally recorded in the step. Doing so makes the time neutral to all the actors within the process. This is referred to as trusted time-stamping. In this way, we are able to demonstrate the *when* of a step.

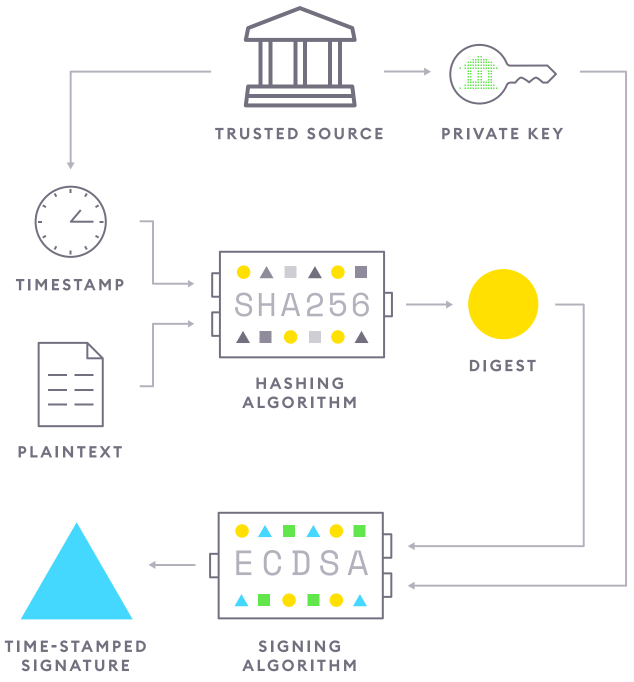

Fig. 6: Trusted Time-Stamping. The proof system obtains the current time from a trusted source and uses that timestamp along with the plaintext document to generate a cryptographic digest, which is immediately signed by the trusted source (using its private key) to create a time-stamped signature.

Contextual Proof

So far, we have covered methods to establish the what (with a cryptographic digest), the who (a digital signature), the when (trusted timestamp) — now we need to establish the where, or the context of a step relative to the others.

We do this by establishing contextual (and cumulative) proof using a hash chain.

First, we create another digest for the entire step (the digest, signature, and timestamp, so far), which we can call a link hash.

Then we insert that link hash into the following step of the process along with the digests of the other key factual elements.

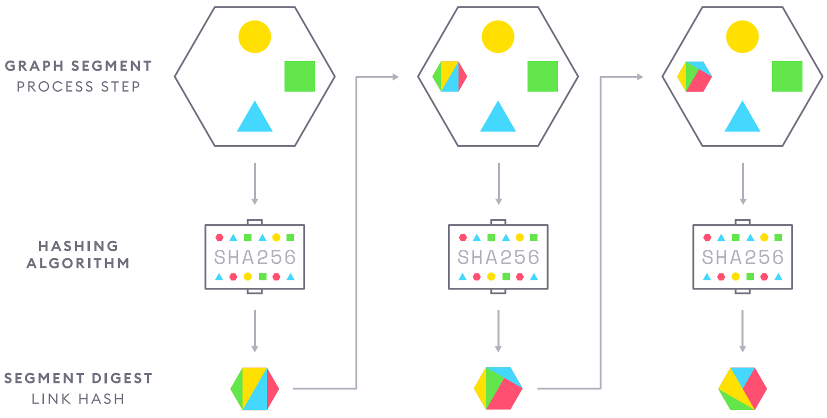

Fig. 7: Contextual Proof via Hash Chain. The digest of the content of each step is included in the following step, cryptographically linking every step of the process. By including the link hash of the previous step, we establish a clear and provable step sequence, and thus proof of the context of all steps. With the exception of the first, a new step simply cannot be created if there is not a previous one to reference.

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 chain of cumulative proofs to ensure the contextual proof.

By sharing a cumulative proof of a specific step, we share the proof of all its previous steps as well. Alice can just carry the digest of the final step — which is effectively proof of all the steps of her Battleship game.

Cumulative proofs are tamper-proof — to change the result of one, you would have to go all the way back to the first step to change the proofs in all previous steps, which would in turn invalidate the entire process.

In this way, we are able to demonstrate the where of a step.

NESTED PROOFS OF PROCESSES

As the proof of an entire process can be represented by the digest of the final step, we could include the proof of one process inside a step in another proof of process graph. The link hash of the final step of the outer process graph would thus demonstrate both proofs of its steps, as well as proof of the steps of the inner process graph.

In this way, we can seamlessly connect multiple processes. For example, the proof of Alice's Battleship victory could be connected to the proof of her victory in a Backgammon game to establish a reputation profile in a network of board game enthusiasts who wish to assert their refined taste in an era of rapid-fire, deliberately addicting console and mobile games.

DEFINING ANTERIORITY AND CONTEXT: COMMON TIME

In this paper we have used examples of individual sources of trust for timestamping. However, the factual element of when (and thus, where) can also be established through consensus on a common timeline.

A common timeline can be established through a blockchain network in which participants agree to contribute computational power through a clearly defined consensus logic for the acceptance of new blocks (process steps) of information that define the when and where of a proof of process graph.

Read more about blockchain consensus logic mechanisms: bitcoin whitepaper, proof of work, proof of stake.

Building the Proof of Process

STEP 1: INITIAL STATE

At the beginning of the game, we store the digest of the position of the player's ships, both players' signatures, and the trusted timestamp. This is the first step of the process graph. The actual data behind the digest — the ship's positions — can be stored by each player separately however they prefer.

STEP 2: ALICE GUESSES

Alice makes her first guess: she calls out "D10." We record the digest of the string "D10," timestamp it, sign it, and link it to the first step. Thus the second step contains:

1. The digest of the string: "D10"
2. Alice's timestamped signature of the digest 3. Alice's public key
3. The timestamp authority's public key
4. The link hash of the first step

STEP 3: BOB RESPONDS

Bob responds "it's a hit!" He puts a red pin on top of the ship at D10 of his primary grid, indicating that Alice scored a hit, and Alice places a pin in the same position on her tracking grid. We record the digest of the string "hit" along with the time when Bob responded, and obtain signatures from Bob for digest and the source of trust for the timestamp. Thus in the third step, we save:

1. The digest of the string "hit"
2. Bob's timestamped signature of the digest 3. Bob's public key
3. The timestamp authority's public key
4. The link hash of the second step

GOING FORWARD

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

In the final couplet, when Alice makes her winning move, Bob responds with “hit.” We store the details for that couplet, and additionally, we record an extra step as the final step which contains:

1. The digest of the string “endgame”
2. Both Alice and Bob’s timestamped signatures of that digest
3. Alice’s public key
4. Bob’s public key
5. The timestamp authority’s public key
6. The link hash of the previous step (where Bob replied with “hit”)

The proof of process graph of all the steps must be recorded by a computing platform common to both the players in a way that each time a player guesses or responds, it can compute the proof. Computing the proof for each step involves hashing the guess/response, getting the players to sign their respective guesses/responses, time-stamping the guess/response, and connecting every new step in the graph with the link hash of the previous step.

The computing platform can publish the hash of the final step of “endgame,” which acts a proof for the entire game, without the need to share any other proof. Each step of the entire process graph can be uniquely referenced through the final link hash, so participants on a network of board game enthusiasts only need to share final hash with others as a proof of the game’s history.

Zero Knowledge Proof of Process

Now that we have the proof of process (demonstrating Alice's victory) decoupled from the actual process (the private record of Alice's guesses and responses), there remains a need to publish the process graph of cumulative proofs in a way that the proof can be publicly verifiable — without revealing the source data behind the proofs. This way, everyone can credibly believe Alice's refined taste and skill in gaming without any knowledge of her unique ship placements and guessing strategies that secured her victories.

To enable this hypothetical reputation network of gamers with taste, 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 integrity of each game being played. However, to make sure that any set of two players do not make up fake games for reputation, or that gamers don't make up fake players to set up easy victories, we need to introduce referees sitting in each game. These could be professional Battleship referees, or simply gamers not currently playing who want to earn good karma and get a foot in the door of the competitive Battleship scene.

If there is a referee attesting each move of a game, and thus each step of a process graph, then we can use a "two out of three signings" approach to establish the truthfulness 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 that a neutral 3rd party attested the game's events.

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 in which the parties are hierarchically organized, we do not always need a third party.

KYC – KNOW YOUR CUSTOMER

Let's imagine that Alice has been so successful at playing Battleship that she now must open a bank account because she is running out of space to store her winnings in her Manhattan studio apartment. She naturally chooses the bank that has been running a campaign advertising their implementation of Proof of Process.

Before she can deposit her winnings, however, she must first go through a mandatory "Know Your Customer" (KYC) process. As part of this government-mandated process, Alice must share her private documents (e.g. birth certificate) with the bank to prove her identity and financial well-being. In this situation the bank acts as the arbiter of Alice's identity, financial health, and other pieces of information. Therefore the bank is also the source of trust.

Thus, just two parties — Alice and the bank — are sufficient to generate a proof of process in the context of a process within a hierarchical system.

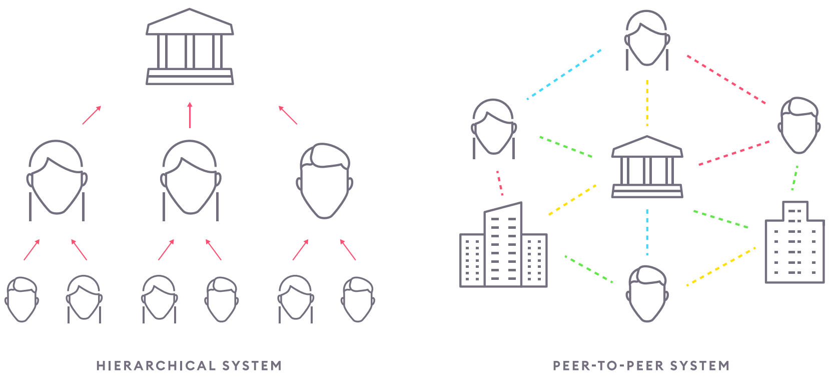

Fig. 8: Hierarchical and Peer-to-Peer Systems. Most organizations function through hierarchical structures in which multiple parties are grouped under the same head, whereas in peer-to-peer systems, information is shared sideways and directly between its members, making the exchange often considerably faster. However, in peer-to-peer systems, trust becomes more challenging to ascertain without a single source of trust like that of the head a hierarchical system.

The bank validates each of Alice's documents and attests them through steps in the proof of process graph with its signature, and then adds one more step: Alice's customer status is verified and her bank account is open.

Finally, Alice can deposit her winnings — and additionally, she receives a special token from the bank! It contains the digest of the final step of her KYC proof of process graph. And it's a good thing too, because this bank has a monthly deposit limit and she can only deposit half of her cash from the Battleship winnings. She's asked her friend Bob to take care of her plants in her apartment this month while she's out of town and is not sure it's a good idea to keep all that cash around. Bob's new career as a gardener hasn't yielded the riches of the Battleship game, and she thinks he might be tempted pad his gardening fees.

Luckily, there's another Proof of Process-enabled bank around the corner, and she is able to skip the time consuming KYC process and immediately open an account with them just by presenting the token — the digest from the first bank. She is able to deposit the rest of her winnings and confidently give Bob the keys to her apartment while she is away for the international Battleship championship. She can rest easy knowing her money and plants are in good hands.

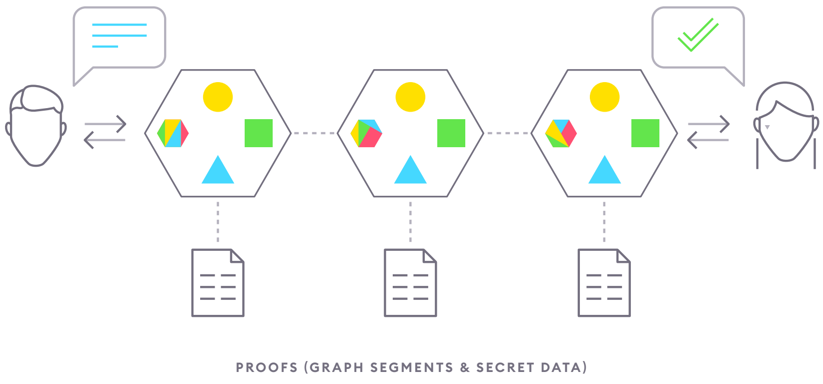

Fig. 9: Zero-Knowledge Proof. If we do not store any secret behind the proof, but only the key factual elements to enable the proof (those being the digest, signature, public key and the trusted timestamp) while still ensuring that the proof can be demonstrated to any verifier, then we have a zero-knowledge proof of process.

With this example, Alice is able to prove her integrity established in the first bank's KYC process to the second bank without having to reveal her private documents again, and with the "two of three signings" Battleship example, she is able to prove the results of her games without having to reveal her secret guesses and ship positions. These are examples of zero-knowledge proofs of process in a hierarchical and peer-to-peer system, respectively.

Proof of process does not necessitate, but it does provide a framework to enable zero-knowledge proof. In this way, we can enable individual privacy on the protocol level.

Through Proof of Process, both peer-to-peer and hierarchical systems can interact and exchange information in a way that each system can utilize its own source(s) of trust, thus enabling a decoupled and modular trust network.

Conclusion

Let's recount the steps that we've taken to create a proof of process.

- Extract trust by deriving proofs of the four key factual elements to address the four information security concerns for each step of the process:
  What: data integrity through cryptographic hashing
  Who: actor non-repudiation through digital signatures
  When: proof of anteriority through trusted time-stamping or common time
  Where: proof of context through cumulative proof via hash chain
- Generate a single proof for each step
- Publish the final proof in a distributed fashion through a network in which truth is established through a consensus mechanism

We've covered the first two steps in some detail while the third has been implied in our examples of tasteful gamers and technologically progressive banks. The fact remains that proofs are only useful if people accept them through usable networks. Proof of Process only functions in a situation of human collaboration and technical implementation.

This human reality is also the fifth factual element that does not belong to any step in a process, but to the entirety of Proof of Process: *why*.

The previous four key factual elements enable the conditions to address the factual element of *why* through *Proof of Applicability*: Why does this proof apply to me? Using this element, the proof must be established in the interpersonal human context of trust — through a network of trust.

Because people are naturally curious about why things are the way they are and most developed countries have abundant access to relatively inexpensive computational power and connectivity, this fifth element is also why people have a strong incentive to build and participate in networks through this protocol. PoP serves as a powerful lens to discover the truth behind situations, as traceability, compliance, privacy, and accountability have been enabled at the protocol level.

In today's hyperconnected world, there are no islands, only continuous movements of information, ideas, conversations, goods, and products. There must be a strong foundation of trust to make this movement possible. Through the Proof of Process protocol it is possible to establish networks that manage trust in a decoupled and modular way, and to thus create a new paradigm for communication, collaboration, exchange, regulation, and governance.

–

This paper was written by the Stratumn team in collaboration with their customers and partners, and published May 10, 2017. Its authors are Anuj Das Gupta, Richard Caetano, Akbar Ali Ansari, Stephan Florquin and Gordon Cieplak.