

Quantstamp: A proposal for automated security audits in the blockchain

Quantstamp is the first decentralized smart contract security-audit platform. We are building the first token-based security-audit infrastructure that enables smart contract developers to improve the security of their programs. Our team is made of up of experts in software testing.

Founders

Richard Ma, Cornell ECE
Algorithmic Portfolio Manager

Steven Stewart, MCS, BA
PhD-candidate, U. Waterloo
Software verification, Database implementation

Founding Team Members

Dr. Vajih Montaghani, PhD
Formal methods

Ed Zulkoski, B.S.
PhD-candidate, U. Waterloo
Mathematics and Computer Science

Advisors

Dr. Vijay Ganesh, Assistant Professor, U. Waterloo
(Ex-Stanford, MIT)

Dr. Derek Rayside, P. Eng., Associate Professor, U.
Waterloo (Ex-MIT)

Introduction	4
The Problem	4
Overview of the QSP	4
Why create a decentralized security platform?	5
How we improve the blockchain infrastructure	5
How we improve the developer's process	6
Quantstamp, by example	6
Team and objectives	7
Why we should be concerned about smart contracts	9
The DAO and others	9
Recent studies	10
Advancing the blockchain infrastructure	13
Decentralized Verification	13
Incremental releases and the subscription model	14
Independent verification	15
Security Disclosure Strategy	15
Architectural View	17
Quantstamp API for Ethereum	17
Quantstamp Network for Ethereum	18
Quantstamp Reports	18
Token Distribution	20
FAQ	20
The Quantstamp Presale	20
Token Allocations	21
Off-chain Tools for Developers	23
Smart Debugging using discriminating examples	23
Developer's View	25
The Security Audit Engine	27
Tradecraft	27
Computer-aided reasoning tools	27
Model-checking	28
Static program analysis	28
Symbolic execution and Concolic Testing	28
Recent contributions by our team	28
Existing Verification and Security Tools	30
Security checks for Ethereum/Solidity	31

Demo: Locating The Parity Multisig Vulnerability	34
Production Phases	35
Financial Planning	36
Frequently Asked Questions	37
Detailed Bios	40
Important Legal Disclaimer	43

Introduction

Quantstamp is the *first* decentralized smart contract security-audit platform, designed to systematically find vulnerabilities in smart contracts. These vulnerabilities threaten the long-term adoption of blockchain technology and cryptocurrencies. We believe that issues of usability and trust of these emergent technologies cannot be strengthened without first addressing security, and that the alternative outcome will lead to the “wild west” of digital crime.

The Quantstamp (QSP) platform is based on blockchain technology for decentralized consensus that maintains a ledger and a history of security audits and reports. The reports are generated whenever an audit of a smart contract is requested. The Quantstamp network (QN) is made up of verifier nodes that perform the security audits and store the results in the decentralized database.

The Problem

Blockchain technology is designed to solve a number of well-known weaknesses of the internet. In *Blockchain Revolution*, Tapscott¹ notes that by requiring all participants to use cryptography, the consequences of reckless behaviour are limited to the *individual* who behaves recklessly, and that this redirection of responsibility potentially solves wide-ranging problems ranging from hacking to spam.

While this idealistic view of blockchain technology is popular among enthusiasts, maintaining this view requires ignoring the increasing number of high profile exploitations² of security vulnerabilities in smart contracts. The unfortunate reality is this: once participants are able to deploy and execute arbitrary computer code where cryptography no longer comes to the rescue, all of the usual weaknesses introduced by human error are ripe for exploitation.

Overview of the QSP

Quantstamp intervenes by providing a critical piece of infrastructure for trust: **a decentralized security platform for smart contracts**. Whereas all participants have previously been required to use cryptography for protecting their digital assets, all software developers will now be asked to submit their code through an automated security audit and crowdsourced bounty system via independent verifiers. The security audits are mechanized, precise, and generate public security reports, serving as a kind of firewall for the blockchain.

¹ Don and Alex Tapscott. *Blockchain Revolution: How the technology behind Bitcoin is changing money, business, and the world*, Portfolio / Penguin Random House Canada, p. 39, 2017.

² We encourage the reader to enter “ethereum and theft” in a Google search.

For the non-expert and expert user alike, trust becomes increasingly quantifiable; one need only evaluate a smart contract's security report via **qsscan.io**³ and observe whether or not it meets their expectations. This functionality ensures that whenever projects update their code, end-users are able to re-verify that no known vulnerabilities were introduced by the changes. The platform rewards verifier-nodes in exchange for running security audits.

Quantstamp ensures that all audited smart contracts conform to a security standard. There are thousands of new projects being released, and Quantstamp's primary goal is to protect user funds from hacks. Although there are no absolute guarantees that a smart contract will be free of all possible vulnerabilities, we can *at least* rule out the most commonly known ones and provide users with a detailed security report. This report enables the developer to correct problems prior to deployment and, once deployed, the report is forever linked to the smart contract. When the developer puts a large enough bounty on the smart contract, the platform crowdsources black/white hat hackers (independent verifiers) to perform human code audits in order to compete for the token bounty.

Why create a decentralized security platform?

One could argue that it is not necessary to build a decentralized security platform. Why not simply have a developer run checks and verify security properties on his own? Or, better yet, why not hire a trusted consultant whose credentials inspire confidence and whose reputation is on the line? Although these are reasonable and complementary ideas, we think this goes against the very principles upon which the blockchain is intended; both options require placing trust in a third-party, which is the antithesis of the decentralized ideal. Instead, we are automating security audits as well as harnessing human intelligence and ensuring that the results are verified by decentralized consensus.

How we improve the blockchain infrastructure

We improve the blockchain infrastructure by implementing a decentralized security protocol. Our security library automates certain security checks and verifies properties, and does so in a trustless⁴ manner. Our approach offers the following two core advantages.

1. The decentralized design allows end-users to directly verify programs without relying on reputation

The most important contribution of blockchain technology is the decentralization of trust based on incentive. Verified smart contracts are tagged with the version of the security library used by the verifier and a security report is published. Independent verifiers are incentivized to scan the

³ The website qsscan.io provides an interface for reviewing security reports of smart contracts.

⁴ We use the word "trustless" to indicate that it is not necessary to trust a third-party.

network for uncaught vulnerabilities, and developers are responsible to address vulnerabilities when they are found.

2. We incentivize miners by making the verification and certification of smart contracts part of the mathematical problem that a miner needs to solve

In a blockchain architecture, “miners” are participating entities that try to add transactions to the chain. In the QSP, miners are called verifiers. A verifier needs to solve a mathematical problem to add a new block of transactions to the open ledger. This problem is hard to solve, but verifying the correctness of a proposed solution by a verifier is very easy. At the same time, the hardness of the problem and the bounty for solving it makes verifiers honest and discourages them from cheating. Our approach is to make the verification and certification of smart contracts a part of the mathematical problem that a miner needs to solve. A verifier that certifies a contract produces a proof that is easily verifiable and in turn, s/he is awarded a bounty. In case a verifier finds a violation of security goals by a contract, s/he produces a counterexample that is a witness to the violation and the entity that has created the contract pays a bounty fee to the first verifier to do so.

How we improve the developer’s process

Well-intentioned software developers need help to produce better code. As pointed out by Luu et al.⁵, there is a semantic gap rooted in a misunderstanding of how code executes in the blockchain; consequently, there is a pressing need for better tools that can assist the developer in capturing vulnerabilities prior to deployment. The current way developers test code, manually, via open source code reviews and unit tests (if they are diligent), is not sufficient to meet the needs of blockchain technology, which ideally offers perfect security. All of the above methods are really manual methods that allow for human error. There is a need for tools that offer as seamless and automatic an experience as possible, while minimizing the chance of serious vulnerabilities slipping through the cracks.

Quantstamp, by example

The utility of QSP is best illustrated by example. Suppose a developer plans to deploy a smart contract written in Solidity on Ethereum. There is substantial risk when writing code that accesses a monetary system, and the developer must be careful to ensure that no funds are lost due to vulnerabilities.

⁵ Luu et al. describe this semantic gap in their paper “Making Smart Contracts Smarter.” They propose to enhance the operational semantics of Ethereum and offer a symbolic execution tool called Oyente to find bugs in smart contracts. We pragmatically believe that very few developers, in practice, will ever utilize such tools, just as very few do in the ordinary practice of software engineering.

In order to minimize risk, the developer submits his code for a security audit, and calls the auditing function directly from his wallet sending a QSP token bounty to the QN network. Depending on the security needs of the program, the developer can decide how much bounty to send. Then, the QN broadcasts the audit request and verifiers immediately perform a set of security checks in order to earn the bounty. Upon consensus, the network publishes a security report that summarizes the results. The report classifies issues based on a severity system from 1–10; a 1 is a minor warning, a 10 is a major vulnerability. From that point on, if a serious vulnerability is not immediately detected, the bounty remains until the specified time has elapsed. At the end of the time period, the bounty is returned to the developer who requested the audit.

When requesting an audit, the developer can choose either a public or private security report. Private reports are encrypted using the public key of the smart contract and can be decrypted by the owner/developer. The developer and the public can access a web portal called qsscan.io to review any security report. By using seamless cryptographic hashing, security reports viewed by the public exactly match the audited source code to prevent manipulation of report results.

A developer can perform security audits on a local machine prior to issuing a public audit, but may find that the computational overhead is too high. Verifier nodes in the QN are likely to have greater computational capacity in terms of memory and processing cores than the average developer's machine. In the same way, by aggregating the power of human hackers with a large bounty, the project is able to greatly surpass the coverage of a standard code review. Once the code is ready for deployment, the developer is ultimately motivated to produce a public security report in order to give users the reassurance that a decentralized security audit was performed.

When a security report identifies issues found within a smart contract, the developer can publicly annotate the report with feedback. This gives developers the power to indicate false-positives in the report, and the community can validate the annotations.

Quantstamp does not guarantee 100% that the source code is flawless, but it provides a much higher degree of assurance that the code is secure using both automated and crowdsourcing methods. The Quantstamp team commits to continuously engaging in research and development, making regular improvements to the security library. When there are new releases, developers can re-audit their smart contracts, demonstrating their commitment to securing code and increasing public confidence.

Team and objectives

Our team has devoted their careers to helping developers produce more reliable code, representing years of combined research and experience in the discipline of software verification. The opportunity to apply these expertise towards the next generation of the digital revolution is extremely exciting for everyone involved. There is a clear and urgent need for more secure code.

We believe that automated security audits will help developers to deploy provably *better* code that the public can trust without having to write formal specifications that contain more lines of code than the program itself. Our aim is to automate checks and property verification as much as possible. Each of these objectives should contribute to a healthier blockchain ecosystem. We are excited to be distinguishing our project by addressing an infrastructural-level problem.

The remainder of this proposal details why a decentralized security platform is a necessary technological advancement, and provides a high-level architecture of the platform.

Why we should be concerned about smart contracts

There is increasing evidence that a troubling percentage, perhaps greater than 40 percent, of Ethereum smart contracts are vulnerable. It would be difficult to conclude that the remaining smart contracts are safe because they may contain as yet unidentified vulnerabilities. This is not a knock on Ethereum, as it is reasonable to assume that any platform that enables the execution of arbitrary code that accesses the monetary system is at serious risk. The onus is clearly on the developer to “get it right.”

The DAO and others

Code is law. Or so they say.

On June 17, 2016, what is now referred to as The DAO has since become synonymous with perhaps one of the greatest would-be heists of modern times. To the tune of \$55 million, an Ether thief discovered a bug in a smart contract that allowed repeated ATM-like withdrawals. There was no eject button, and once a smart contract is deployed, there’s no turning back. To the attacker’s delight, smart contracts are immutable and publically available for the unscrupulous to study and exploit.

Date	Losses	Description
June 17, 2016	\$55 million	The DAO exploit ⁶ is perhaps the best-known. A non-recursive function could be re-entered before termination, leading to loops of invocations that consume all gas. The unhandled exception meant that repeated withdrawals were possible in the calling function.
June 20, 2017	\$32.6 million	A vulnerability in Parity's multisignature wallet was exploited by hackers ⁷ . In this case, some Solidity primitives have the non-obvious side effect of invoking the fallback function of the recipient. This can lead to unexpected behaviour and may be exploitable by an attacker.
July 31, 2017	\$1 million	There was an error in the smart contract of the REX token sale ⁸ . Specifically, when generating the contract bytes for deployment, a mistake was made defining the constructor parameters. Instead of a quoted string for an address, a Javascript hex string was used. Although this was not a theft by an attacker, it was a preventable loss.

⁶ <https://www.multichain.com/blog/2016/06/smart-contracts-the-dao-implosion/>

⁷ <https://www.cnbc.com/2017/07/20/32-million-worth-of-digital-currency-ether-stolen-by-hackers.html>

⁸ <https://blog.rexmls.com/the-solution-a2eddbda1a5d>

Of course, what followed was the (in)famous and controversial Ethereum hard fork, intended to correct the apparent wrong-doing of the attacker. Perhaps, to the outsider, it's surprising that the hard fork would be controversial; after all, who could condone the actions of the world's greatest thief? But, therein lies the problem: if, in fact, code is law, then should it not be respected for how it was written? Although the developer of the smart contract undoubtedly did not intend to offer an ATM service, the code itself, as written, most certainly *did* permit this behaviour. If code is law, then the code *and* the law permitted the theft and there was no wrongdoing.

Whatever your thoughts are on the code is law question, in our view one thing is certain: never assume that a smart contract is safe. So long as code has access to a monetary system, and so long as human beings want to make money, then no code is ever truly safe. All we can really do is minimize the risk, and even better is when we can provably eliminate certain types of vulnerabilities that are well-known to be exploitable and damaging. While it is true that there does not exist any fully automated solution that can, without a shadow of a doubt, catch all possible bugs in a computer program, we can confidently state that the risk can be greatly minimized. In fact, one could argue that any bug worth finding will tend to be found, and those that are not will tend to not matter.

Still, were there only one incident -- however damaging it was -- then perhaps our worries would be out of proportion. The occasional theft could be absorbed as a kind of nuisance tax, and not necessarily perceived as a catastrophe. (Ho hum another theft. It happens.) Unfortunately, there is no such thing as bug insurance (not yet) and faulty code, when it surfaces, can indeed be catastrophic. Beyond that, it's simply impractical for there to be a hard fork whenever there is a theft.

Of course, finding a bug isn't easy. Even if the bug could self-identify, it would be difficult for an automated solution to be absolutely certain without somehow understanding the original intentions behind the code. Sometimes what looks like a bug is actually a feature! What can we do?

Our response: learn and keep learning. Identify patterns and classes of vulnerabilities. Use established techniques and improve them when necessary. Wrap this all up and make it a part of a security library whose outputs are verified by decentralized consensus. Incentivize contributors, and harness both the power of white and black hat hackers to assist in the effort. Reward them when they succeed. Keep developers accountable.

Recent studies

The full extent to which security vulnerabilities plague smart contracts is unknown; however, recent studies make it abundantly clear that there really is a *plague*. Below, we summarize a

short selection of research papers that characterize some of the most serious vulnerabilities, some of which highlight just how easy it is for developers to unknowingly make mistakes.

<p>Making Smart Contracts Smarter</p> <p>Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. <i>Making Smart Contracts Smarter</i>. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). ACM, New York, NY, USA, 254-269. DOI: https://doi.org/10.1145/2976749.2978309</p>	<p>Both malicious miners and users can exploit certain classes of vulnerabilities that the authors deem to be due to a “semantic gap” between how the developer thinks code executes versus how it actually does. In their study, 8,519 out of 19,366 (44%) Ethereum smart contracts contained “semantic gap” vulnerabilities, involving a total balance of over 6 million ETH⁹.</p>
<p>Formal verification of smart contracts: Short Paper</p> <p>Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, and Santiago Zanella-Béguelin. 2016. <i>Formal Verification of Smart Contracts: Short Paper</i>. In Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security (PLAS '16). ACM, New York, NY, USA, 91-96. DOI: https://doi.org/10.1145/2993600.2993611</p>	<p>The authors translate Solidity to F* to analyze EVM bytecode. They perform checks to capture whether the code undoes side effects that can persist when a call to send() fails, and also to detect the reentrancy problem that plagued The DAO.</p> <p>The limitations of their tool restrict analysis to only 46 smart contracts; however, the authors state that of those only a handful passed their checks, suggesting that “a large-scale analysis of published contracts would likely uncover widespread vulnerabilities.”</p>
<p>Demystifying Incentives in the Consensus Computer</p> <p>Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. 2015. <i>Demystifying Incentives in the Consensus Computer</i>. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). ACM, New York, NY, USA, 706-719. DOI: https://doi.org/10.1145/2810103.2813659</p>	<p>The authors show that Turing-complete scripting exposes miners to a new class of attacks: “Honest miners are vulnerable to attacks in cryptocurrencies where verifying transactions per block requires significant computational resources.” To address this problem, they propose an incentive structure to the consensus protocol where cheating provides no intrinsic advantage.</p>
<p>A survey of attacks on Ethereum smart contracts</p> <p>Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. <i>A Survey of Attacks on Ethereum Smart Contracts</i>. In Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204, Matteo Maffei and Mark Ryan (Eds.), Vol. 10204. Springer-Verlag New York, Inc., New York, NY, USA, 164-186. DOI: https://doi.org/10.1007/978-1-4939-9739-7_10</p>	<p>The authors present a taxonomy of security vulnerabilities observed across the corpus of Ethereum smart contracts. In general, these vulnerabilities emerge due to subtleties of Solidity that are unknown or misunderstood by developers.</p>

⁹ To be precise, the value of 6,169,802 ETH on 2017-July-23 is about \$1.4 billion USD.

https://doi.org/10.1007/978-3-662-54455-6_8	
<p>Step by step towards creating a safe smart contract</p> <p>D. Delmolino et al. <i>Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab</i>. Cryptology ePrint Archive, Report 2015/460, 2015. http://eprint.iacr.org/</p>	<p>The authors show how even a very simple contract for playing Rock, Paper, Scissors can contain several logical flaws. These are characterized as <i>Contracts that do not refund, Lack of cryptography to achieve fairness, Incentive misalignment</i>.</p>
<p>Safer smart contracts through type-driven development</p> <p>J. Pettersson and R. Edström. <i>Safer smart contracts through type-driven development: Using dependent and polymorphic types for safer development of smart contracts</i>. Masters Thesis in Computer Science, Chalmers University of Technology of Gothenburg, Sweden, 2016.</p>	<p>Three classes of errors are highlighted that are common in smart contracts: <i>unexpected states, failure to use cryptography, and full call stack</i>. The authors propose using dependent and polymorphic types and a functional language called Idris to make smart contract development safer.</p>

While the above papers are only a sample, a noteworthy percentage of smart contracts reportedly have known vulnerabilities. Our perspective is that it is possible to prevent many of these by performing automated checks and formally verifying expected properties. While it is likely that some attackers will focus their efforts on high profile, opportunistic heists of large magnitude, many others will be content with multiple smaller grabs less likely to garner much attention. Everybody is at risk.

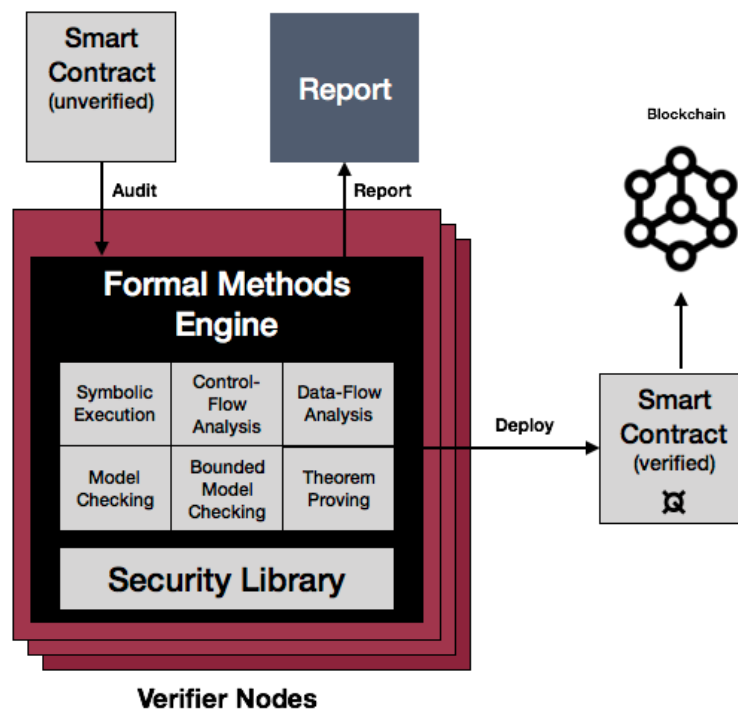
Advancing the blockchain infrastructure

We propose decentralized security audits as a means to inhibit the deployment of vulnerable smart contracts. With the emergence of platforms like Ethereum, a new model for building scalable distributed open-sourced applications has emerged that allows arbitrary smart contract code to be executed without sufficient safeguards in place.

Decentralized Verification

Our decentralized platform for verification rewards participants who provide compute resources for the purpose of running checks on smart contracts as well as independent verifiers who contribute discriminating examples that break the code. These checks make up the **Security Library** and are run by decentralized verifiers in our network. The Quantstamp protocol ensures the verification and certification of smart contracts is part of the mathematical problem that a verifier needs to solve.

When a verifier certifies a contract, a higher margin of confidence is reached. When a violation of security goals is found, the verifier produces a counterexample that is a witness to the violation and the entity that has created the contract pays the bug bounty to the verifier. In exchange for the costs of running the verification computations and providing trust to smart contracts, verifier nodes are paid a computation fee.



The **Security Audit Engine** takes an unverified smart contract as input, performs the automated security and vulnerability checks, and produces a report. The developer then has the option of reviewing any warnings or show stoppers. Verification results will be attached with a version code that shows the scope of checks from that version of the Security Library.

The time taken to run the full tests in the Security Library scales with the complexity of the smart contract code; therefore, verification rewards are proportional to computation time. Verifiers require incentivization to motivate participation in this effort, and a token is issued for users to access its features.

We believe that the smart contract developers will be motivated to use these features because of the increased confidence the public gains when knowing that a smart contract was verified transparently by consensus and by ongoing attempts by independent verifiers to break it to win the bounty. Furthermore, as new vulnerabilities are discovered, the Security Library will evolve and new versions will be released. When that happens, users will be motivated to re-verify their smart contracts using the latest version of the Security Library, ensuring that their code is not open to attack due to newly discovered vulnerabilities. This is similar to how users of software can download patches to fix security vulnerabilities, or how users can update their anti-virus application, except that the security of their funds is at stake instead of the security of their personal computer.

Incremental releases and the subscription model

Software releases for the Security Library will have critical, major and minor update version tags. When developers deploy code, they have the ability to flag the contract for re-verification upon each critical/major/minor release on a subscription payment model. For very financially sensitive contracts, developers can choose re-verification on all releases. For less sensitive contracts, they can choose re-verification only on critical releases. When developers flag the contract for verification, and a subsequent verification fails, they will be notified by the network and can take immediate action.

The market price of the token transaction fee is an essential component of the platform that will balance computational resource supply and recurring demand. Because the market price of the token is free-floating, decentralized verifier nodes are incentivized by market forces to dynamically bring on additional resources to meet demand.

A developer can choose to not subscribe because they are confident in their application and do not want to pay subscription fees, but have a critical vulnerability in the code that is only uncovered at a later date by a new release. There is a possibility that vulnerabilities may be

discovered at a later date in contracts that have already been deployed to the network with an earlier version of the Security Library.

Independent verification

In order to increase the health of the overall ecosystem, verifier nodes are able to perform *independent* verification on existing smart contracts. In the process of an independent verification, human verifiers can use any means at their disposal to break the code, and if a smart contract is found to have major vulnerabilities, then the verifier is awarded the rolling bug bounty that was attached to the contract, for doing work that improves the security of the whole platform.

In open source software, developers are often unrewarded for finding bugs. Recently¹⁰, Emin Gün Sirer found two critical vulnerabilities in BitGo while on vacation, and wrote a friendly email to alert them. In a common experience among security developers, he received a thankless reply and later was actually snubbed by the BitGo CTO. The automated bounty reward payout will allow skilled developers to perform independent verification via their verifier nodes and earn immediate monetary rewards and public recognition without all the back-and-forth with companies.

We believe that it will be possible for skilled developers to earn an income purely via independent verification, by manually searching for security flaws in public smart contracts on our platform. Financially sensitive contracts worth millions of dollars, should in theory have bounty contracts worth at least tens of thousands of dollars before being deployed live. This will increase the security of our platform and also incentivize more security experts to spend time in the ecosystem and develop their skills due to the increased immediate value of their work.

Security Disclosure Strategy

Attackers might choose to leverage the security library as a tool for finding vulnerabilities in existing smart contracts. Any detected vulnerabilities could then be used as a starting point for planning an attack. Clearly, it is not our intention to facilitate the efforts of attackers, no matter how unlikely it is that they would succeed.

In theory, this unfortunate scenario could be avoided from the start if all deployed smart contracts were pre-audited by the QSP without ever providing attackers with access to the complete security library. For this reason, we will take the following actions:

1. We will implement a staging period during the library release process, during which time we will generate encrypted security reports that smart contract owners can access.

¹⁰ <http://hackingdistributed.com/2017/07/20/parity-wallet-not-alone/>

2. We will publish public statistics indicating the frequency with which critical issues are present in smart contracts with the hope of motivating smart contract owners to read security reports and take appropriate actions.
3. To avoid giving clues to would-be attackers, we will ensure that the existence of a report will not be indicative of the existence of a vulnerability, nor will characteristics of the encrypted report, such as its size, offer any reliable clues.

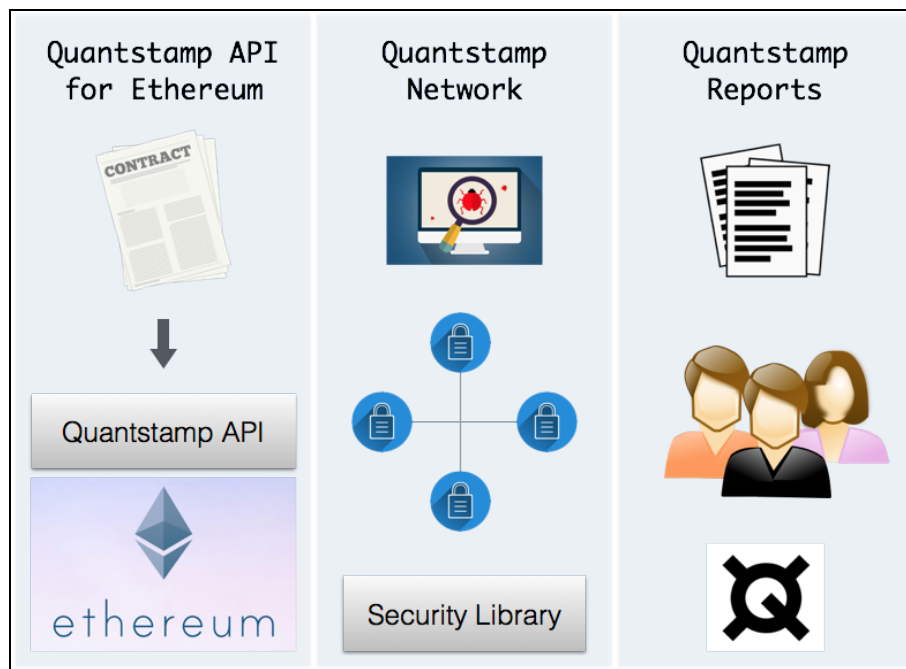
Whenever a new version of the security library is released, there may be a window of time in which previously audited smart contracts have newly detectable vulnerabilities. This again, could give an attacker the opportunity to use the security library as a starting point for planning an attack, even if that window of opportunity is relatively small. This is a secondary purpose of the independent verifier system - by leveraging human intelligence with bounties, we can bridge the gap between inadequate automated checking and the converse - sophisticated automated attacks.

Architectural View

The QSP is split into three conceptual categories:

1. Quantstamp API
2. Quantstamp Network (QN)
3. Quantstamp Reports

The QN is its own independent decentralized network of verifier nodes that maintains the ledger and generates security reports. As a service, the QSP may be thought of as **platform-agnostic**. This is true in the sense that there can be many variations of the security library, one of which includes Solidity (for Ethereum), and variants that may cover other smart contract languages for different platforms. Additionally, the Quantstamp API is a tool that we provide that helps end-users to verify smart contracts.



Quantstamp API for Ethereum

Here, we discuss the Quantstamp API in terms of Ethereum, although the API itself may be implemented for other platforms in the future.

The following list of functions are accessible to the end-user.

register()

All users must first register an Ethereum address by calling this function. This alerts the QSP to monitor the transactions of the registered address.

audit()

A user can send the source code of a smart contract to the QSP to perform a security audit along with a token bounty fee. Upon success of the automated check, the smart contract will be digitally signed to prove that it passes critical security checks. At this point, an encrypted security audit report is made available. The bug bounty remains on the contract to incentivize independent verifiers, until the specified time limit runs out.

deploy()

A user can deploy a smart contract that has passed a security audit.

subscribe()

A user can create a subscription for their smart contract, which ensures that it will be audited whenever new versions of the security library are released. This is advantageous so that newly-discovered vulnerabilities can be detected in previously deployed smart contracts.

upgrade()

Upgrade an existing smart contract. The new version of the smart contract must pass a security audit. Existing bounties are rolled forward.

Quantstamp Network for Ethereum

The Quantstamp Network (QN) is a specialized decentralized blockchain network capable of monitoring transactions related to a registered smart contract involving calls to the Quantstamp API. Essentially, the network monitors and reacts to function calls to the Quantstamp API on Ethereum as if the functions were called on the QN itself.

The QN is primarily responsible for performing security audits and confirming the results of those audits by decentralized consensus. From this, security reports are generated, and all security reports (encrypted or unencrypted) are published as part of the shared history of the network.

Quantstamp Reports

Quantstamp Reports provide a public view of the “transactions” of the QN. Mainly, these transactions contain reports generated by calls to the Quantstamp API for Ethereum. These reports will be made visible via a web-based user-interface at qsscan.io.

Reports can be either public or private. Public reports are visible to everyone in a human-readable form. Private reports are encrypted using the public key of the registered user.

This implies that only the registered user can read the contents of the report. Once a smart contract is deployed, the final security audit report of the smart contract is public. This allows investors and other users of the smart contract to assess the security report themselves before sending Ether.

Additionally, smart contract owners are encouraged to annotate the security report. Although critical vulnerabilities will block the deployment of the smart contract via the Quantstamp API, lesser vulnerabilities (or those that may be prone to false positives) will not block the deployment of the smart contract, but will be published as part of the public security report.

Owners are encouraged to indicate a response to all issued security warnings and flagged issues. The response may be as simple as “this is a false positive” or “we are not concerned about this issue,” or may be highly detailed. The onus is on the owner to provide as much information as possible to anyone who may want to read the security report in order to increase the level of trust. A “trust score” will be computed for each smart contract based upon a combination of the findings in the security report, the size of the bounty, the length of time the bounty has been active and feedback from the community.

Token Distribution

FAQ

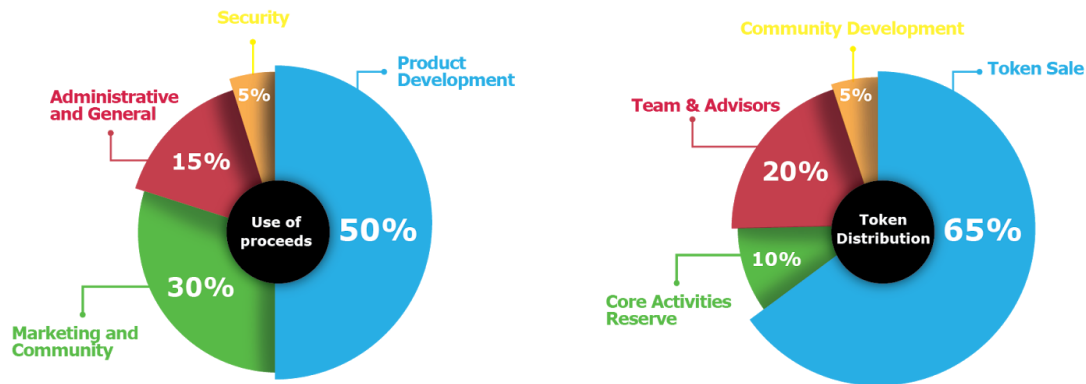
- **100%** of contributions go towards the developing the Quantstamp protocol
- Presale cap: **\$3 million**
- There are no individual caps for the presale, early adopters allow the **hiring of engineers**
- For the main sale, there are **individual caps** and partnering with a leading KYC provider
- Unsold tokens are burned
- Minimum contribution is 0.1 Ether, to reduce unnecessary network load
- Tokens for the founding team are linearly **vested** on a 12 month schedule

The Quantstamp Presale

Token QSP	Type Ethereum ERC20 Token
Price (Includes 100% bonus) 10,000 QSP = 1 ETH	Issuance Balance can be checked immediately. Transfers will be enabled at the end of the main token sale after security auditing.
Sale Period Currently taking pre-orders Sep 29th 6pm PST - Oct 29th 6pm PST	Presale Bonus Bonus starts at 100% in the first week and decreases by 10% each week

The github code for our token distribution: <https://github.com/quantstamp/token-distribution>

Token Allocations



Role of Token	Enable security audits on the Quantstamp Network
Symbol	QSP
Supply	1,000,000,000 QSP
For Sale	650,000,000 QSP
Emission Rate	No new tokens will be created
Price	5,000 QSP = 1 ETH
Accepted Currencies	ETH
Sale Period	November 2017
Token Distribution Date	7 days following security audits at the end of the token sale
Minimum Goal	\$3 million
Maximum Goal	\$30 million

Additional information regarding the token distribution is in the [FAQ section](#) of our white-paper.

Off-chain Tools for Developers

In addition to the decentralized security platform, we will provide a set of off-chain tools aimed at simplifying the development, debugging, and deployment of smart contracts via the Quantstamp API. This includes the application of recent work by one of our team members into creating smarter debugging tools.

Below, we describe the notion of “smart debugging,” that we hope to apply in the context of smart contracts. We then show what the development procedure will look like from the point-of-view of a developer.

Smart Debugging using discriminating examples

Software models with mathematical or logical foundations have proven valuable to software engineering practice by enabling software engineers to focus on essential abstractions, while eliding less important details of their software design. Like any human-created artifact, a model might have imperfections at certain stages of the design process: it might have internal inconsistencies, or it might not properly express the engineer’s design intentions.

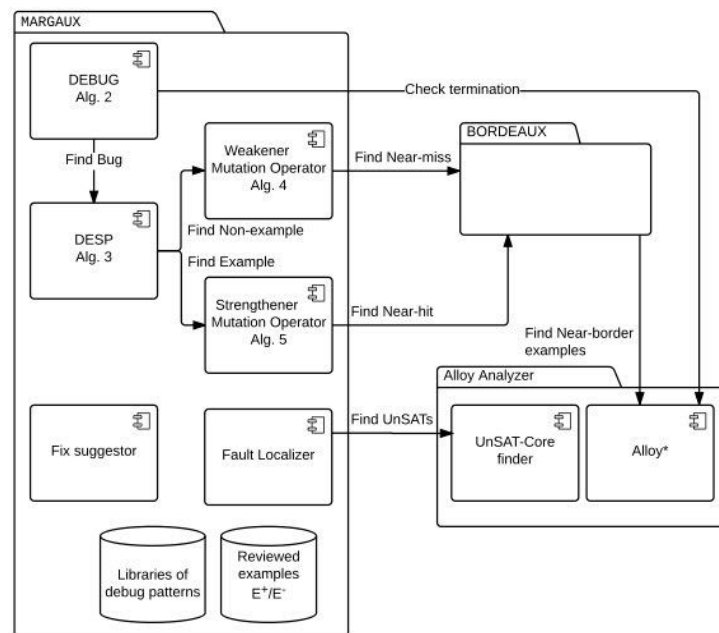
We introduce the idea of a smart debugger that helps a non-expert developer to find flaws and vulnerabilities based on the proven localization, understanding, and fix strategy. This work is explored in depth in the dissertation *Debugging Relational Declarative Models with Discriminating Examples* by founding team member Vajih Montaghani and PhD supervisor Dr. Derek Rayside (University of Waterloo).

The need to debug arises because the *expressed* meaning model differs from the *intended* meaning, but the user does not know where or why. Debugging can be a cumbersome and time-consuming task that persists throughout the software lifecycle. Zeller¹¹, in his seminal book on debugging imperative programs, evokes an inspiring image: *Some people are true debugging gurus. They look at the code and point their finger at the screen and tell you: “Did you try X?” You try X and voila!, the failure is gone.* What has the debugging guru done? They have

¹¹ A. Zeller. Why programs fail: a guide to systematic debugging. Morgan Kaufmann, 2009.

identified, localized, and corrected the bug¹², and they have done this by first forming a hypothesis.

We have developed tools and techniques to provide some automated support for this vision in the context of relational logic models for software abstractions. Two such tools are called Bordeaux and Margaux (depicted in the architectural diagram below). These tools first help the user identify and understand the bug by forming a hypothesis about what might be wrong with the model and computing a discriminating example for the user to accept or reject. If the user judges that a bug has been identified, then further automated analysis helps localize which



part of the model needs to change, and might provide a high-level conceptual description of the correction (but the user still needs to make the correction by hand).

Examples, like test-cases for programs, are more valuable if they reveal a discrepancy between the expressed model and the engineer’s design intentions. We propose the idea of discriminating examples for this purpose. A discriminating example is synthesized from a combination of the engineer’s expressed model and a machine-generated hypothesis of the engineer’s true intentions. A discriminating example either satisfies the model but not the hypothesis, or

¹² A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck. The state of the art in end-user software engineering. *ACM Computing Surveys*, 43(3):21:1–21:44, Apr. 2011., and J. F. Krems. Expert strategies in debugging: experimental results and a computational model. In *Cognition and Computer Programming*, pages 241–254. Ablex Publishing Corp., 1994.

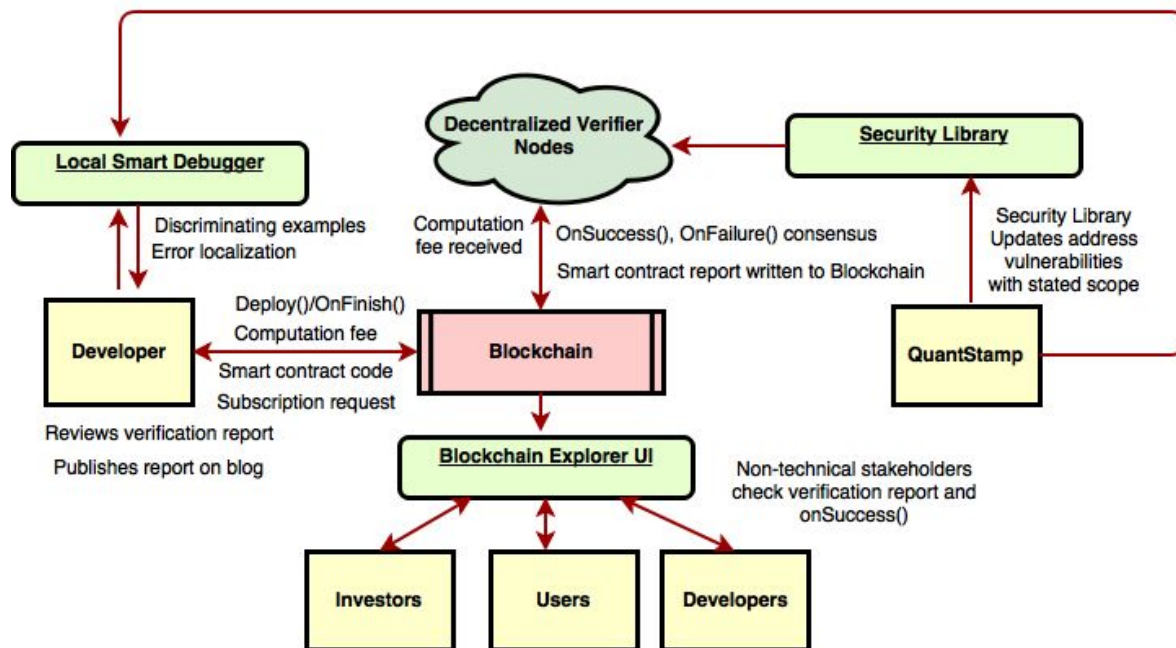
satisfies the hypothesis but not the model. It shows the difference between the model and the hypothesized alternative.

Validating that the model is a true expression of the engineer's intent is an important and difficult problem. One of the key challenges is that there is typically no other written artifact to compare the model to: the engineer's intention is a mental object. One successful approach to this challenge has been automated example-generation tools, such as the Alloy Analyzer. These tools produce examples (satisfying valuations of the model) for the engineer to accept or reject. These examples, along with the engineer's judgment of them, serve as crucial written artifacts of the engineer's true intentions.

We believe that smart debugging¹³ can ease the burden on the developer, who often struggles to recognize gaps between what he intends the code to do versus what it really does. A smart debugger enables the developer, who likely lacks training in formal methods, to apply localization, understanding, and fixing of bugs.

Developer's View

Workflow from developer's point-of-view.



¹³ The team behind *Agrello*, whose ongoing ICO fundraiser as of 2017-July-22 has reached \$15M USD, claim to use artificial intelligence to infer the intentions of authors of legal contracts. In a sense, that is also an example of smart debugging; however, we doubt their AI claim. A practical smart debugger can guide the human intellect towards bridging semantic gaps with the use of discriminating examples to correct flaws in logical reasoning and supply automatic error localization tools.

1. Write a smart contract.
2. Inspect code for problems. Apply corrections as needed.
3. Push the smart contract to a verifier. If a flaw is found, go back to Step 2. Otherwise, continue to Step 4.
4. Deploy the smart contract along with a subscription tag and a token bounty. Verifier nodes receive the smart contract and proceed to verify it. If a flaw is found, a report can be evaluated. If it is a major flaw, the bounty is paid out. Otherwise, the smart contract is signed with a version number, indicating that it has been verified.
5. Verifiers must agree by consensus protocol that the smart contract is verified. When consensus is reached, the smart contract is considered “deployed”; otherwise, return to Step 2.
6. When the subscription tag is triggered, the contract is re-verified. The developer pays a fee from their subscription wallet and receives a report.

The Security Audit Engine

The Security Audit Engine (SAE) builds upon a tradecraft of tools and techniques founded upon the study of discrete mathematics, logic, and computer science. It interacts with the security library component of the decentralized verification architecture, which maintains a list of checks (to be performed) and properties (to be verified). These are performed automatically in a decentralized manner.

Tradecraft

In real world practice, peer reviewing and unit testing are the major software verification techniques in use. While peer review is an effective approach, it is still prone to human error and manual testing is always limited in coverage and scope. Software verification using automated reasoning tools can help close the gap. Although research into automated reasoning tools started several decades ago, their practical importance has progressed rapidly in the last few years.

We summarize the tradecraft that supports the SAE below.

Computer-aided reasoning tools

Computer-aided reasoning tools, such as SAT/SMT solvers (below), have had a dramatic impact on software engineering and security in recent years. The key reason for the adoption of solvers in software engineering is the continuous improvement in their performance and expressive power.

SAT solvers

SAT (satisfiability) solvers support software verification tools. Computer programs are modeled as Boolean formulas, which are passed to the solver. When modeling program behaviour and testing for particular conditions, a Boolean formula can be constructed such that the existence of a satisfying assignment signifies the presence of a bug. A SAT solver reports “satisfiable” if it can find a solution or, if none exists, reports “unsatisfiable.”

SAT-solvers are important tools in several areas of software engineering, including software verification, program analysis, program synthesis, and automatic testing. Additional applications span a variety of problem domains that include electronic design automation, computer-aided design, and others. SAT-solvers are surprisingly efficient, combining decision heuristics, deductive reasoning, and various experimentally validated techniques.

SMT solvers

An SMT solver is a tool that decides satisfiability of formulas in combination of various first-order theories. It is a generalization of a SAT solver and can handle richer theories than propositional logic. Common first-order theories, which can model fragments of computer code

for vulnerability analysis, include equality, bit vectors, arrays, rationals, integers, and difference logic. This is a very active research area, and there are many applications: software verification, programming languages, test case generation, planning and scheduling, and more. Well known SMT solvers include Yices (SRI), Z3 (Microsoft), CVC3 (NYU, Iowa), STP (Stanford), MathSAT (U. Trento, Italy), Barcelogic (Catalonia, Spain).

Model-checking

Model checking is based on abstracting on the behavior of code in an unambiguous manner, which often leads to the discovery of inconsistencies. This technique explores all possible system states in a brute-force manner.

In contrast to model-checking, *bounded* model-checking (BMC) is a technique for verifying that a given property (typically expressed as an assertion by a user) holds for a program in the number of loop iterations and recursive calls bounded by a given number k , placing a bound on the size of the execution path for finding a bug. This problem can be reduced to solving the Boolean satisfiability problem using SAT-solvers.

The utility of bounded model-checking is in part supported by the *small-scope hypothesis*. This hypothesis states that most bugs have small counterexamples, and has proven to be an effective idea for finding bugs in software models. This hypothesis is the basis for so-called *lightweight* formal methods.

Static program analysis

Static analysis determines properties of a program without actually executing the program. Automated tools can assist programmers and developers in carrying out static analysis. Static analysis has been used to find potential null pointer bugs and to verify that device drivers always respect API usage requirements. (Later in this document, we apply an example static analysis technique to the Parity Multisig vulnerability.)

Symbolic execution and Concolic Testing

Concolic testing is a hybrid software verification technique that performs symbolic execution, a classical technique that treats program variables as symbolic variables, along a concrete execution path. Symbolic execution is used with an automated theorem prover to generate new test cases. Its main focus is finding bugs rather than proving correctness.

Recent contributions by our team

The following table comprises a partial selection of formal methods and software verification projects connected to our combined research efforts. When necessary, we will be adapting these

proven tools and techniques, and others, towards achieving decentralized verification in the blockchain and improving the debugging process for smart contract developers.

Name	Contributors (alphabetical order)	Description
Alloy and the Alloy Analyzer	Vajih Montaghmi Derek Rayside Steven Stewart	Alloy is a relational logic that enables developers to model and reason about software abstractions. The Alloy Analyzer is capable of mechanically generating examples of a user's model. It was originally developed at MIT as part of the Software Design Group under the guidance of Dr. Daniel Jackson. http://alloy.mit.edu/alloy/
Bordeaux	Derek Rayside	Bordeaux is a technique and extension of Alloy for producing near-border examples, an important capability for improving debugging for identifying partial over-constraint bugs in software models. https://github.com/drayside/bordeaux
Clafer	Ed Zulkoski	Clafer is a general-purpose lightweight modeling language developed at GSD Lab, University of Waterloo and MODELS group at IT University of Copenhagen. Lightweight modeling aims at improving the understanding of the problem domain in the early stages of software development and determining the requirements with fewer defects. Clafer's goal is to make modeling more accessible to a wider range of users and domains. http://www.clafer.org/
Margaux	Derek Rayside Vajih Montaghmi	Margaux is a tool for pattern-based debugging that can guide a user to find a bug. The github page includes an architectural diagram for how a debugger using discriminating examples can guide developers towards correcting flaws in logical reasoning. https://github.com/vmontagh/margaux
MapleSAT MapleCOMSPS MapleGlucose	Vijay Ganesh Ed Zulkoski	The award-winning Maple series are a family of conflict-driven clause-learning SAT solvers developed at the University of Waterloo under the supervision of Dr. Vijay Ganesh. https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/

MathCheck	Vijay Ganesh Ed Zulkoski	A constraint programming system that combines SAT solvers with computer-algebra systems. Extended known results on two conjectures related to hypercubes. https://sites.google.com/site/uwmathcheck/
Miramichi	Derek Rayside Steven Stewart	Miramichi is an experimental parallel SAT-solver that leverages GPUs for performance acceleration. https://bitbucket.org/sstewart2015/miramichi4j
Moolloy	Derek Rayside Steven Stewart	Moolloy is an extension to a relational logic for expressing discrete multiobjective optimization problems, with applications in science, software engineering, and finance. https://github.com/TeamAmalgam/moolloy
Petitcodiac	Derek Rayside Steven Stewart	Petitcodiac is an experimental solver for quantifier-free linear real arithmetic (LRA) that leverages OpenMP and GPUs. SMT-solvers, such as Yices and Microsoft's Z3, typically use a variation of the simplex procedure also employed by Petitcodiac. https://github.com/sstewart2012/peticodiac
STP	Vijay Ganesh	STP is a constraint solver (or SMT solver) aimed at solving constraints of bitvectors and arrays. These types of constraints can be generated by program analysis tools, theorem provers, automated bug finders, cryptographic attack tools, intelligent fuzzers, model checkers, and by many other applications. https://github.com/stp/stp

Existing Verification and Security Tools

The blockchain implementation of Nick Szabo's idea¹⁴ of a smart contract is a computer program whose correct execution is enforced without relying on a trusted authority. The popularity of this idea has been growing since the advent of Ethereum. Unfortunately, security bugs in smart contracts enable attackers to steal funds, and the pseudo-anonymity property of the blockchain, which although has mainly positive advantages, means that attackers can operate with minimal risk of repercussions.

¹⁴ Formalizing and Securing Relationships on Public Networks:
<http://firstmonday.org/ojs/index.php/fm/article/view/548/469>

In this section, we briefly highlight existing tools and approaches to improving smart contract and blockchain implementations from the point of view of software verification. Where appropriate, we can compare and contrast the existing verification and security tools with the Quantstamp approach.

Security checks for Ethereum/Solidity

The Ethereum protocol supports stateful contracts, meaning that the values of state variables persist across multiple invocations. A contract is invoked when it receives transactions from users at its unique address. If such transactions are accepted by the blockchain, all participants of the mining network execute the contract code. The network then agrees, by the consensus protocol, on the output and next state of the contract. Given that Ethereum smart contracts are immutable and the effects of the transactions cannot be reversed, it is clearly essential to be able to reason effectively about code prior to deployment.

Atzei et al. describe a taxonomy of vulnerabilities and unexpected behaviours of smart contracts written in Solidity for Ethereum. Although this taxonomy is specific to Ethereum, it is likely that similar vulnerabilities will exist for other platforms that use contracts in the future. We summarize this taxonomy below based on their findings.

Call to the unknown	Some Solidity primitives have the non-obvious side effect of invoking the fallback function of the recipient. This can lead to unexpected behaviour and may be exploitable by an attacker. (We discuss this in the section on the Parity/Multisig vulnerability.)
Exception disorder	There are two different behaviours for how exceptions are handled that depend on how contracts call each other. For some, side effects of the whole transaction are reverted; for others, only the side effects of the invocation of another smart contract are reverted. These irregularities can affect the security of contracts.
Gasless send	When a user sends ether to a contract, it is possible to incur an out of gas exception.
Type casts	The compiler can do some type-checking, but there are circumstances where types are not checked which can lead to unexpected behaviour.
Reentrancy	The fallback mechanism may allow a non-recursive function to be re-entered before its termination, which could lead to loops of invocations that consume all gas. (The “DAO attack” infamously exploited this vulnerability.)

Keeping secrets	Declaring a field as private does not guarantee its secrecy because the blockchain is public and the contents of a transaction are inspectable. Cryptographic techniques may need to be employed to protect secrets.
Immutable bugs	Deployed contracts cannot be altered, including when they have bugs, and there is no direct way to patch it. (An exception to this occurred after the DAO attack when a controversial hard fork of the blockchain nullified the effects of transactions involved in the attack.)
Ether lost in transfer	Ether sent to orphaned addresses is lost forever, and there is no way to detect when an address is an orphan.
Stack size limit	The call stack is bounded by 1024 frames and a further invocation triggers an exception. (A hard fork of the Ethereum blockchain in October 2016 has addressed this vulnerability.)
Unpredictable state	The state of a contract upon sending a transaction to the network is not guaranteed to be the state of the contract when it actually executes. Additionally, miners are not required to preserve the order of transactions when grouping them into a block. Attackers can exploit this “transaction-order dependence” vulnerability.
Generating randomness	A malicious miner can craft his block to bias the outcome of pseudo-random generator number in his favor. For example, this could be advantageous for lotteries, games, etc.
Time constraints	Many applications use time constraints to determine which actions are permitted in the current state. If a miner holds a stake on a contract, he could gain an advantage by choosing a suitable timestamp for a block he is mining.

Below are a sample of checks that would be implemented in the Security Library for Solidity.

Constant functions	The compiler does not enforce that a constant method is not modifying state; instead, this should be enforced.
Contracts that receive ether directly	Contracts that receive Ether directly need to implement a fallback function in order to receive Ether, otherwise the function throws an exception and sends back the Ether. There can be an alert when the fallback function is not implemented, since there are situations where the programmer would want to do this.
Fallback function	A contract can have exactly one fallback function, and it cannot spend more than 2300 gas. We can automatically test that the

	programmer is spending less than 2300 gas inside that fallback function.
Reentrancy exploit	<p>When calling another contract, the called contract can change state variables of the calling contract via its functions. It's possible to check that calls to external functions happen after changes to state variables in the current contract so that it is not vulnerable to a reentrancy exploit.</p> <p>https://gist.github.com/chriseth/c4a53f201cd17fc3dd5f8ddea2aa3ff9</p>
Implicit declaration	A variable declared anywhere within a function will be in scope for the entire function, regardless of where it is declared. It is also initialized to a default value for the entire scope of the function. It is possible that poorly written code can access an implicitly declared variable with a default value. When this happens, our report would generate an alert.
Transaction owner	When checking tx.origin, it gets the original address that kicked off the transaction. A malicious actor can use an attack wallet to drain all funds if the smart contract code required tx.origin == owner, since in this case tx.origin would be the address of the attack wallet.
Gas forwarding	There is an extremely dangerous feature called addr.call.value(x)() that can forward gas to a receiving contract and opens up the ability to perform more expensive actions. This is a problem that needs to be explored more in-depth later.

Demo: Locating The Parity Multisig Vulnerability

We provide a demonstration of a generalizable technique for automatically locating vulnerabilities similar to the Parity Multisig Wallet flaw that lead to a \$32.6 million theft.

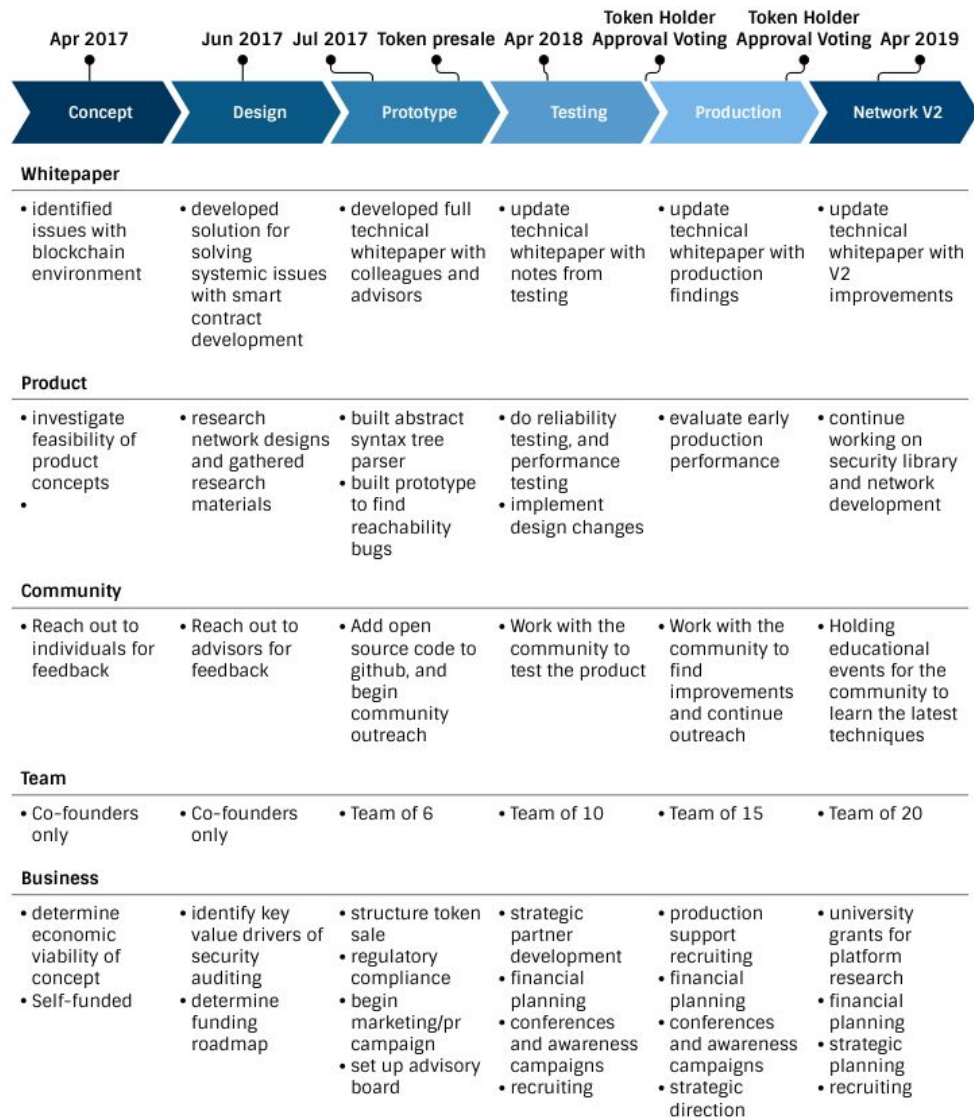
This simple analyzer constructs multiple AST (Abstract Syntax Tree) visitors and uses these to extract the program variables and call structure of a Solidity contract. The analyser finds any public method that directly or indirectly exposes a non-public state variable modification, and alerts the developer. Using call-graphs we can capture a class of vulnerabilities that can be located as solutions to reachability¹⁵ problems. In this demo, we have two example solidity contracts to show how the analyser identifies a direct and an indirect vulnerability.

Github code for the demo: <https://github.com/quantstamp/solidity-analyzer>

¹⁵ <https://en.wikipedia.org/wiki/Reachability>

Production Phases

Quantstamp Production Phases



Financial Planning



Frequently Asked Questions

Q. What is Quantstamp?

Quantstamp is a decentralized security platform for smart contracts that tokenizes security audits and report generation. The advantages of a decentralized security platform include automation and trust.

Q. What is the Quantstamp team going to deliver?

The Quantstamp team will be developing the following:

1. A decentralized blockchain platform called the Quantstamp Network that monitors Ethereum transactions for registered users
2. A security library for Solidity
3. An implementation of the Quantstamp API for Ethereum

A security library may also be developed to support languages other than Solidity, and the Quantstamp API may be implemented for other platforms. In this respect, the design is modular and platform-agnostic.

Q. Aren't human security audits and code reviews the state-of-the-art?

To begin, let's first acknowledge an unfortunate truth: software developers write buggy software and, by and large, cannot be trusted to "get it right." While seemingly a harsh critique, writing correct, bug-free software is very difficult. (Every seasoned developer eventually comes around to this conclusion.) In particular, the existence of software bugs is anticipated and accepted when developing most popular applications, and perhaps most disconcerting from the security perspective is that many visionaries are predicting that the majority of these applications will run on the blockchain in the future.

One member of our team noted, anecdotally, that at a previous software company (a profitable one at that), the backlog of bugs was in the hundreds, and the project manager was constantly juggling a list of 20-30 features and bugs to work on in every 2-week sprint, struggling to make any significant progress. In spite of an abundance of bugs, customers expressed satisfaction about the product and mainly only reacted strongly when "show-stoppers" were uncovered. Unfortunately, once you give programmers access to a monetary system via smart contracts, just about any bug can be a show-stopper.

To improve software, most developers believe that they merely need to conduct more code reviews and write more unit tests, but the cost/benefit calculation seldom favours increased

testing. Although reliance on unit testing and code reviews may be acceptable for low-risk applications, it is not acceptable when writing code for critical systems. Instead, computer chip manufacturers, airplane and automobile manufacturers, and many others rely on automated software verification to complement other best-practices. For similar reasons, our approach is to take advantage of the years of research that have developed these sophisticated techniques.

Q. Can I really trust a computer to find vulnerabilities better than a human can on his own?

While it is true that unit testing and code reviews go a long ways towards improving the quality of software, it has been shown that techniques based on formal methods are better at finding the most subtle and critical bugs that evade human inspection. This is true, in large part, because of the ability of automated reasoning tools to simulate critical execution paths in a manner that well exceeds the limitations of human cognition.

Another way to look at this is to consider what has transpired in recent years in algorithmic trading. For years, it was believed that humans were better at trading than computers, until eventually the computers took over¹⁶. With a quick online search for “computers have taken over Wall Street,” you’ll find numerous articles on this phenomenon.

Perhaps, not surprisingly, something similar is already underway with automated security audits: maybe, when we start, we cannot match an experienced human except on the cost/speed tradeoff, but with each new release the automated solution will be able to catch more and more security issues in a transparent way until eventually the algorithms will beat humans.

In the meantime, we leverage human intelligence via an automated bounty for bugs that are found by independent verifiers (white hat hackers).

Q. Why tokenize security audits? Instead, why not form a security consulting company?

One could argue that it is not necessary to build a decentralized security platform. Why not simply have a developer run checks and verify security properties on his own? Or, better yet, why not hire a trusted consultant whose credentials inspire confidence and whose reputation is on the line? Although these are reasonable ideas, we think this goes against the very principles upon which the blockchain is intended; both options require placing trust in a human third-party, which is the antithesis of the decentralized solution. Instead, we are taking a systematic approach, ensuring conclusions about security are verified by decentralized consensus.

¹⁶ The Quants Are Taking Over Wall Street:
<https://www.forbes.com/sites/nathanvardi/2016/08/17/the-quants-are-taking-over-wall-street/>

In fact, one of the first things we considered was creating a consulting company, but there are several drawbacks to that method. For example, while security consulting companies already exist, major and minor vulnerabilities still exist in about half of all smart contracts. The other issue with consulting companies is that their methods are not transparent. Instead of trusting the original programmer, the end-user is just trusting the consulting company instead. The third issue with a consulting company is that the cost barrier is very high, which discourages widespread usage.

We concluded that a systematic solution made more sense. It had to hit the three points above, *i.e.*, transparency, ease of verifiability, low cost. Just about the only solution that allows for these three characteristics is the decentralized approach. By using multiple verifiers through the blockchain, and opening up the security library, and paying via a token, our work would become just enhancing the security library and publishing knowledge about security instead of hand-holding each participant in a manual/high-cost way like with a consulting company. We're also capable of servicing a much larger number of projects (essentially all the projects); anyone that wants to search their smart contracts for flaws can do so at a low cost (maybe 5-20 cents compared to thousands of dollars for a consulting company).

One great feature of the network is *independent verification*. Since a lot of work in blockchains are coming from open-source developers, who may not want to pay fees but can be sloppy with their coding, this feature is really great for the health of the open-source community, and it's really only possible by putting our security library on the blockchain and paying these independent verifiers with new tokens.

Q. Why not use Why3 or similar tool for formal verification instead?

Existing projects such as Why3 are too inaccessible for the typical smart contract developer to use. A similar argument can be made about the adoption of alternative programming paradigms, such as functional programming (OCaml, Haskell, Clojure), where there ends up being a lot of hype and promise but, upon closer inspection, not a lot of adoption by actual developers, who still prefer Java, C#, C++, and Python. For all these reasons and more, Quantstamp automates as much of the security auditing process as possible by embedding it into the decentralized consensus protocol, and relieving the developer from having to learn specialized techniques.

Detailed Bios

Co-founders



Steven Stewart

University of Waterloo ECE, Software Verification

Steven is a PhD candidate at the University of Waterloo (ECE) where, under Derek Rayside and Krzysztof Czarnecki, he focuses on improving the performance of software verification tools and solvers using distributed computing and GPUs.

Previously, Steven co-founded a San Francisco-based startup called Many Trees Inc that used GPUs for machine learning and Big Data analytics. In his spare time, he likes tinkering with in-memory databases accelerated using GPUs. He spent nearly 5 years as part of Canada's cryptologic agency in the Department of National Defense.



Richard Ma

Cornell ECE, Algorithmic Portfolio Manager

Algorithmic Portfolio Manager at Bitcoin HFT Fund. Ex-Tower Research Capital Quant Strategist. Programmed production algorithmic trading software in C++/Python/R on competitive US, European, and Asian derivatives exchanges. Wrote tens of thousands unit tests and built production-grade integration and validation testing software. Due to Richard's extreme testing and risk-management methodology, his HFT trading systems had zero notable incidents in nearly a decade of reliably handling millions of dollars of investor capital.

Founding team members



Dr. Vajih Montaghami, PhD
Formal Methods

Vajih Montaghami received his PhD from the University of Waterloo for his work on verifying and debugging lightweight formal models. He focused on different aspects of software engineering artifacts including declarative software model formal analysis, programming language static analysis, imperative code systemization, and software architecture analysis and evaluation.

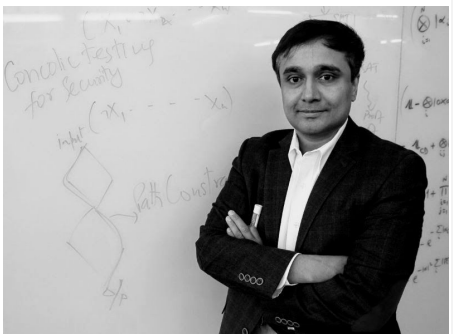
During his PhD study, he worked at Google and experienced dealing with large-scale data analysis systems. At Google he worked on automating end-to-end testing of a machine learning algorithm applied to a massive data source. More recently, at Amazon, Vajih helped develop highly scalable systems as a backend software engineer.



Ed Zulkoski,
B.S., Mathematics and Computer Science

Edward Zulkoski is a Ph.D. candidate in the Department of Computer Science at the University of Waterloo under the supervision of Vijay Ganesh and Krzysztof Czarnecki. He recently completed an internship at Microsoft Research under the direction of Dr. Christopher Wintersteiger. His PhD research is focused on studying and exploiting the structural properties of SAT and SMT formulas. His earlier work investigated combinations of SAT solvers with computer algebra systems, and optimization techniques for multi-objective product line optimization. Ed was awarded a Ph.D. Fellowship from IBM Canada's Centers for Advanced Studies Research.


Advisors



Dr. Vijay Ganesh,
Assistant Professor, University of Waterloo

Dr. Vijay Ganesh is an assistant professor at the University of Waterloo. Prior to that, he was a research scientist at MIT, and completed his PhD in computer science from Stanford University in 2007. Vijay's primary area of research is the theory and practice of automated reasoning aimed at software engineering, formal methods, security, and mathematics.

Vijay has won numerous awards, most recently the ACM Test of Time Award at CCS 2016, the Early Researcher Award in 2016, Outstanding Paper Award at ACSAC 2016, an IBM Research Faculty Award in 2015, two Google Research Faculty Awards in 2013 and 2011, and a Ten-Year

	<p>Most Influential paper award at DATE 2008. In total, he has won 9 best paper awards/honors.</p>
	<p>Dr. Derek Rayside, P. Eng, Associate Professor, University of Waterloo</p> <p>Derek Rayside is an Associate Professor of Electrical & Computer Engineering at the University of Waterloo. His primary research areas are lightweight formal methods and program analysis. He received his doctorate in Computer Science at MIT.</p> <p>Derek is an advisor to a Waterloo startup that was recently acquired by Microsoft.</p>

Important Legal Disclaimer

Quantstamp Technologies Inc. (the “**Company**” or “**Quantstamp**”) Tokens (the “**Tokens**” or “**QSP Tokens**”) to be offered at the Quantstamp Token Pre-Sale and the Public Sale (collectively, the “**Token Sale**”) are not intended to constitute securities in any jurisdiction. This document (the “**White Paper**”) does not constitute a prospectus or offer document of any sort and is not intended to constitute an offer of securities or a solicitation for investment in securities in any jurisdiction.

This White Paper does not constitute or form part of any opinion on any advice to sell or any solicitation of any offer by Quantstamp to purchase any QSP Tokens, nor shall it or any part of it, nor the fact of its presentation form the basis of, or be relied upon in connection with, any contract or investment decision.

No person is bound to enter into any contract or binding legal commitment in relation to the sale and purchase of the QSP Tokens and no cryptocurrency or other form of payment is to be accepted on the basis of this Whitepaper.

Any agreement between Quantstamp and you as a purchaser in relation to any sale or purchase of QSP Tokens is to be governed a separate Quantstamp Token Sale Terms and Conditions document (the “**Terms**”). In the event of any inconsistencies between the Terms and this Whitepaper, the former shall prevail.

You are not eligible and you are not to purchase any QSP Tokens in the Quantstamp Token Sale if you are a citizen, resident (for tax purposes or otherwise) or green card holder of the United States of America or a citizen of the People’s Republic of China.

No regulatory authority has examined or approved of any of the information set out in this Whitepaper. No such action has been or will be taken under the laws, regulatory requirements or rules of any jurisdiction. The publication, distribution or dissemination of this Whitepaper

does not imply that the applicable laws, regulatory requirements or rules have been complied with.

There are risks and uncertainties associated with Quantstamp and its business and operations, the QSP Tokens and the Quantstamp Token Sale.

This Whitepaper, any part thereof and any copy thereof must not be taken or transmitted to any country where distribution or dissemination of this Whitepaper is prohibited or restricted.

CLOSED SYSTEM UTILITY

As of the date of publication of this paper, the Tokens have no known potential uses outside of the Quantstamp ecosystem, and are not permitted to be sold or otherwise traded on third-party exchanges. This paper does not constitute advice nor a recommendation by Quantstamp, its officers, directors, managers, employees, agents, advisors or consultants, or any other person to any recipient of this paper on the merits of the participation in the Token Sale. Quantstamp Tokens should not be acquired for speculative or investment purposes with the expectation of making a profit or immediate re-sale. No promises of future performance or value are or will be made with respect to Quantstamp Tokens. Accordingly, no promise of inherent value, no promise of continuing payments, and no guarantee that Quantstamp Tokens will hold any particular value is made. Unless prospective participants fully understand and accept the nature of Quantstamp and the potential risks inherent in Quantstamp Tokens, they should not participate in the Token Sale.

Quantstamp Tokens are sold as a functional good and all proceeds received by Quantstamp may be spent freely by Quantstamp absent any conditions, save as set out herein.

DISCLAIMER OF LIABILITY

To the maximum extent permitted by the applicable laws, regulations and rules, Quantstamp shall not be liable for any indirect, special, incidental, consequential or other losses of any kind, in tort, contract or otherwise (including but not limited to loss of revenue, income or profits, and loss of use or data), arising out of or in connection with any acceptance of or reliance on this Whitepaper or any part thereof by you.

NO REPRESENTATIONS AND WARRANTIES

Quantstamp does not make or purport to make, and hereby disclaims, any representation, warranty or undertaking in any form whatsoever to any entity or person, including any representation, warranty or undertaking in relation to the truth, accuracy and completeness of any of the information set out in this Whitepaper.

In particular, no representations or warranties whatsoever are made with respect to Quantstamp or the Tokens:

- (a) merchantability, suitability or fitness for any particular purpose;
- (b) that the contents of this document are accurate and free from any error(s);
- (c) that such contents do not infringe any third party rights. Quantstamp shall have no liability for damages of any kind arising out of the use, reference to, or reliance on the contents of this document, even if advised of the possibility of such damages;

This Whitepaper references third party data and industry publications. Quantstamp believes that these references are accurate; however, Quantstamp does not provide any assurances as to the accuracy or completeness of this data. We have not independently verified the data sourced from third party sources in this paper, or ascertained the underlying assumptions relied upon by such sources.

REPRESENTATIONS AND WARRANTIES BY YOU

By accessing and/or accepting possession of any information in this Whitepaper or such part thereof, you represent and warrant to Quantstamp as follows:

- (a) you acknowledge that the QSP Tokens do not constitute securities in any form in any jurisdiction;

(b) you acknowledge that this White Paper does not constitute a prospectus or offer document of any sort and is not intended to constitute an offer of securities in any jurisdiction or a solicitation for investment in securities and you are not bound to enter into any contract or binding legal commitment and no cryptocurrency or other form of payment is to be accepted on the basis of this Whitepaper;

(c) you acknowledge that no regulatory authority has examined or approved of the information set out in this Whitepaper, no action has been or will be taken under the laws, regulatory requirements or rules of any jurisdiction and the publication, distribution or dissemination of this Whitepaper to you does not imply that the applicable laws, regulatory requirements or rules have been complied with;

(d) you agree and acknowledge that this Whitepaper, the undertaking and/or the completion of the Quantstamp Token Sale, or future trading of the QSP Tokens on any cryptocurrency exchange, shall not be construed, interpreted or deemed by you as an indication of the merits of Quantstamp, the QSP Tokens and the Quantstamp Token Sale;

(e) the distribution or dissemination of this Whitepaper, any part thereof or any copy thereof, or acceptance of the same by you, is not prohibited or restricted by the applicable laws, regulations or rules in your jurisdiction, and where any restrictions in relation to possession are applicable, you have observed and complied with all such restrictions at your own expense and without liability to Quantstamp;

(f) you agree and acknowledge that in the event that you wish to purchase any QSP Tokens, the QSP Tokens are not to be construed, interpreted, classified or treated as:

(i) any kind of currency other than cryptocurrency;

(ii) debentures, stocks or shares issued by any person or entity;

(iii) rights, options or derivatives in respect of such debentures, stocks or shares;

- (iv) rights under a contract for differences or under any other contract the purpose or pretended purpose of which is to secure a profit or avoid a loss;
- (v) units in a collective investment scheme;
- (vi) units in a business trust;
- (vii) derivatives of units in a business trust; or
- (viii) any other security or class of securities.

(g) you are fully aware of and understand that you are not eligible to purchase any QSP Tokens if you are a citizen, resident (tax or otherwise) or green card holder of the United States of America or a citizen or resident of the Republic of Singapore;

(h) you have a basic degree of understanding of the operation, functionality, usage, storage, transmission mechanisms and other material characteristics of cryptocurrencies, blockchain-based software systems, cryptocurrency wallets or other related token storage mechanisms, blockchain technology and smart contract technology;

(i) you are fully aware and understand that in the case where you wish to purchase any QSP Tokens, there are risks associated with Quantstamp and its business and operations and the Tokens;

(j) you agree and acknowledge that Quantstamp is not liable for any indirect, special, incidental, consequential or other losses of any kind, in tort, contract or otherwise (including but not limited to loss of revenue, income or profits, and loss of use or data), arising out of or in connection with any acceptance of or reliance on this Whitepaper or any part thereof by you; and

(k) all of the above representations and warranties are true, complete, accurate and non-misleading from the time of your access to and/or acceptance of possession of this Whitepaper or such part thereof.

CAUTIONARY NOTE ON FORWARD-LOOKING STATEMENTS

All statements contained in this Whitepaper, statements made in press releases or in any place accessible by the public and oral statements that may be made by Quantstamp's respective directors, executive officers, employees or other representatives acting on behalf of Quantstamp that are not statements of historical fact, constitute "forward- looking statements". Some of these statements can be identified by forward-looking terms such as "aim", "target", "anticipate", "believe", "could", "estimate", "expect", "if", "intend", "may", "plan", "possible", "probable", "project", "should", "would", "will" or other similar terms. However, these terms are not the exclusive means of identifying forward-looking statements. All statements regarding Quantstamp's financial position, business strategies, plans and prospects and the future prospects of the industry which Quantstamp is in are forward-looking statements. These forward-looking statements, including but not limited to statements as to Quantstamp's revenue and profitability, prospects, future plans, other expected industry trends and other matters discussed in this Whitepaper regarding Quantstamp are matters that are not historical facts, but only predictions.

These forward-looking statements involve known and unknown risks, uncertainties and other factors that may cause the actual future results, performance or achievements of Quantstamp to be materially different from any future results, performance or achievements expected, expressed or implied by such forward-looking statements. These factors include, amongst others:

- (a) changes in political, social, economic and stock or cryptocurrency market conditions, and the regulatory environment in the countries in which Quantstamp conducts its respective businesses and operations;
- (b) the risk that Quantstamp may be unable to execute or implement its business strategies and future plans;
- (c) changes in interest rates and exchange rates of fiat currencies and cryptocurrencies;
- (d) changes in the anticipated growth strategies and expected internal growth of Quantstamp;
- (e) changes in the availability and fees payable to Quantstamp in connection with its respective businesses and operations;

- (f) changes in the availability and salaries of employees who are required by Quantstamp to operate their respective businesses and operations;
- (g) changes in competitive conditions under which Quantstamp operates, and the ability of Quantstamp to compete under such conditions;
- (h) changes in the future capital needs of Quantstamp and the availability of financing and capital to fund such needs;
- (i) war or acts of international or domestic terrorism;
- (j) occurrences of catastrophic events, natural disasters and acts of God that affect the business and/or operations of Quantstamp;
- (k) other factors beyond the control of Quantstamp; and
- (l) any risk or uncertainties associated with Quantstamp and its businesses and operations and the QSP Tokens.

All forward-looking statements made by or attributable to Quantstamp or persons acting on behalf of Quantstamp are expressly qualified in their entirety by the factors listed above. Given the risks and uncertainties that may cause the actual future results, performance or achievements of Quantstamp to be materially different from that expected, expressed or implied by the forward-looking statements in this Whitepaper, undue reliance must not be placed on these statements. These forward-looking statements are applicable only as of the date of this Whitepaper.

Neither Quantstamp, nor any other person represents, warrants and/or undertakes that the actual future results, performance or achievements of Quantstamp will be as discussed in those forward-looking statements. The actual results, performance or achievements of Quantstamp may differ materially from those anticipated in these forward- looking statements.

Nothing contained in this Whitepaper is or may be relied upon as a promise, representation or undertaking as to the future performance or policies of Quantstamp.

Further, Quantstamp disclaims any responsibility to update any of those forward-looking statements or publicly announce any revisions to those forward-looking statements to reflect

future developments, events or circumstances, even if new information becomes available or other events occur in the future.

MARKET AND INDUSTRY INFORMATION AND NO CONSENT OF OTHER PERSONS

This Whitepaper includes market and industry information and forecasts that have been obtained from internal surveys, reports and studies, where appropriate, as well as market research, publicly available information and industry publications. Such surveys, reports, studies, market research, publicly available information and publications generally state that the information that they contain has been obtained from sources believed to be reliable, but there can be no assurance as to the accuracy or completeness of such included information.

Save for Quantstamp and its directors, executive officers and employees, no person has provided his or her consent to the inclusion of his or her name and/or other information attributed or perceived to be attributed to such person in connection therewith in this Whitepaper and no representation, warranty or undertaking is or purported to be provided as to the accuracy or completeness of such information by such person and such persons shall not be obliged to provide any updates on the same.

While Quantstamp has taken reasonable actions to ensure that the information is extracted accurately and in its proper context, Quantstamp has not conducted any independent review of the information extracted from third party sources, verified the accuracy or completeness of such information or ascertained the underlying economic assumptions relied upon therein. Consequently, neither Quantstamp nor its respective directors, executive officers and employees acting on their behalf make any representation or warranty as to the accuracy or completeness of such information and shall not be obliged to provide any updates on the same.

TERMS USED

To facilitate a better understanding of the QSP Tokens being offered for purchase Quantstamp, and the business and operations of Quantstamp, certain technical terms and abbreviations, as well as, in certain instances, their descriptions, have been used in this Whitepaper. These

descriptions and assigned meanings should not be treated as being definitive of their meanings and may not correspond to standard industry meanings or usage.

Words importing the singular shall, where applicable, include the plural and vice versa and words importing the masculine gender shall, where applicable, include the feminine and neuter genders and vice versa. References to persons shall include corporations.

NO ADVICE

No information in this Whitepaper should be considered to be business, legal, financial or tax advice regarding Quantstamp, the QSP Tokens and the Quantstamp Token Sale. You should consult your own legal, financial, tax or other professional adviser regarding Quantstamp and its business and operations and the QSP Tokens. You should be aware that you are bearing the financial risk of any purchase of QSP Tokens for an indefinite period of time.

NO FURTHER INFORMATION OR UPDATE

No person has been or is authorised to give any information or representation not contained in this Whitepaper in connection with Quantstamp and their respective businesses and operations, the QSP Tokens and, if given, such information or representation must not be relied upon as having been authorised by or on behalf of Quantstamp. The Quantstamp Token Sale shall not, under any circumstances, constitute a continuing representation or create any suggestion or implication that there has been no change, or development reasonably likely to involve a material change in the affairs, conditions and prospects of Quantstamp or in any statement of fact or information contained in this Whitepaper since the date hereof.

RESTRICTIONS ON DISTRIBUTION AND DISSEMINATION

The distribution or dissemination of this Whitepaper or any part thereof may be prohibited or restricted by the laws, regulatory requirements and rules of any jurisdiction. In the case where any restriction applies, you are to inform yourself about, and to observe, any restrictions which are applicable to your possession of this Whitepaper or such part thereof at your own expense and without liability to Quantstamp.

Persons to whom a copy of this Whitepaper has been distributed or disseminated, provided access to or who otherwise have the Whitepaper in their possession shall not circulate it to any other persons, reproduce or otherwise distribute this Whitepaper or any information contained herein for any purpose whatsoever nor permit or cause the same to occur.

RISKS AND UNCERTAINTIES

Prospective purchasers of QSP Tokens should carefully consider and evaluate all risks and uncertainties associated with Quantstamp, the QSP Tokens, the Quantstamp Token Sale, all information set out in this Whitepaper and the Terms prior to any purchase of QSP Tokens. If any of such risks and uncertainties develops into actual events, the business, financial condition, results of operations and prospects of Quantstamp could be materially and adversely affected. In such cases, you may lose all or part of the value of the QSP Tokens.

IF YOU ARE IN ANY DOUBT AS TO THE ACTION YOU SHOULD TAKE, YOU SHOULD CONSULT YOUR LEGAL, FINANCIAL, TAX OR OTHER PROFESSIONAL ADVISOR(S).