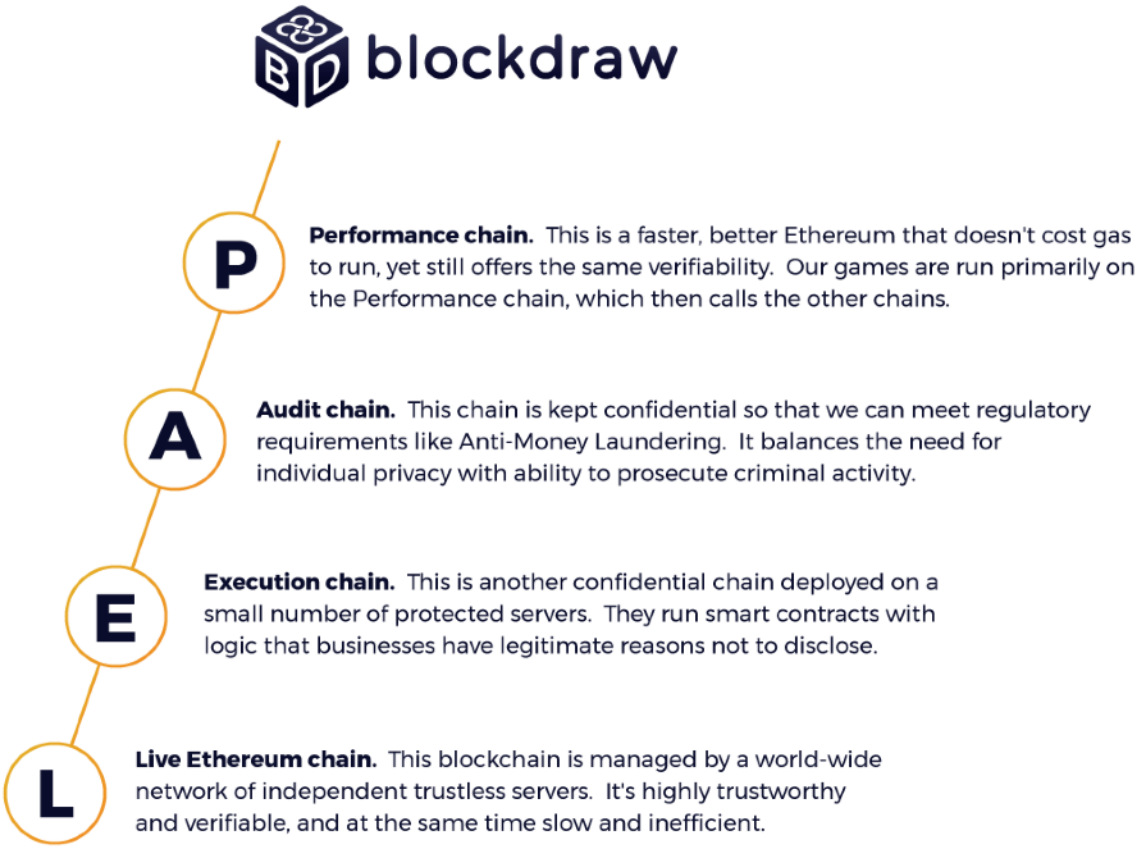
blockdraw

Technology Frequently Asked Questions

FAQ V1.1

Business class security and Integrity combined with the verifiability and trustless capability of the blockchain.



PEA = Blockdraw's three Ethereum forked **Performance**, **Execution**, and **Audit** Blockchains.

HOW DO THE NOTARY STAMPS GUARANTEE THE INTEGRITY OF LEAP?

The hashes used are from the Secure Hash Algorithm (SHA) family, which are widely considered to be one-way hashes. That means knowing the output (the hash) we can't figure out what input (the block) was used to create it. In fact, no one has been able to find a pattern among even sets of inputs and outputs, which means the best method is to randomly guess. Since the hashes themselves are 256 bits

long, a brute force search is on the order of 2^256. This number is astronomically large; we'd have to turn every atom in the observable universe into a computer to start getting close. Thus, once we've made a Notary stamp public, it can be considered an irrevocable commitment to the prior contents of the issuing chain. It's simply too hard to forge.

HOW DOES USING MULTIPLE CHAINS INCREASE INTEGRITY OVER A SINGLE BLOCKCHAIN?

We Notary stamp each of the LEAP chains with one another, in addition to the Live chain. Every time we stamp, that means everything prior to the stamp on that one chain gets graven in stone. And each time we stamp to the Live chain, that provides public commitment that verifies we can't change that one chain. But it turns out the system of chains eventually provides that same guarantee, even if there were no Live chain. After enough transactions back and forth, it will happen that two stamps on two different chains will "cover" each other. That makes everything they mutually cover infeasible to forge. Since the chains are constantly interacting, the stamps keep increasing coverage until all early blocks on all chains can't be changed any more. We call this method cross-hashing notarization, and its proof of correctness is that same as for Directed
Acyclic Graphs.

HOW CAN WE TRUST YOUR EXECUTION IF YOU DON'T PUBLICALLY REVEAL YOUR SMART CONTRACTS?

Publically revealing smart contracts enables open certification; anyone can look at the code and verify it works properly. It turns out that we don't need open certification to ensure trustless cooperation, we only need individual verifiability.

Being trustless means that if any participant tries to take advantage of another by cheating, they can always be caught by another participant. So, the answer is, you don't need to trust us; we'll give you all the open source tools needed to verify for yourself that everything we do, and every player does, is above board. Moreover, there's a mathematical proof called the finite state equivalence theorem where if our code meets certain criteria (strongly connected et al) we can show it always plays fairly, without needing to reveal the code. We plan on building a tool whereby anyone can certify code meets the proof requirements without ever seeing our actual code. Thus, two businesses could interact with one another fairly and securely, each assured the other is going to honor their contracts, without either needing to know the details of the other's implementation. Our goal is selective confidentiality, not secrecy. Our accounting smart contracts are on the Live chain, and all the calls on the Performance chain are publically viewable, so we disclose quite a bit already. The only thing kept confidential is the precise logic used to run the game or business transaction, and that's both modular best practices in programming and the way companies prefer to operate.

I UNDERSTAND HOW NOTARY STAMPS GUARANTEE CHAIN INTEGRITY, BUT HOW DO WE KNOW YOU CAN'T FORGE DATA IN BETWEEN STAMPS?

We return to the idea of trustlessness in the previous question. The PEA chains are so easy to create that it might appear straightforward for a rogue employee to forge a fake chain. But since every player independently has sufficient information to verify the chain and a lot of game play happens within our decentralized apps which are notary stamped to our readable performance chain, that ruse would quickly be discovered. We can estimate how fast it would happen. We notary stamp to the live chain twice per user session, so if there are 1000 concurrent users playing an hour apiece, on average we'll be stamping once every two seconds. That's well below Ethereums block generation time! At that scale it becomes

mathematically infeasible to forge a chain. Further note, that as our system scales and gets more users, the notary stamp interval becomes even smaller, significantly increasing integrity. So as more people use LEAP, it becomes more difficult for anyone to spoof it, including, you guessed it, whoever is running the servers!

HOW DOES THE GOLLE ALGORITHM WORK WITH LEAP TO PROVIDE AIRTIGHT VERIFIABILITY?

During the key steps of gameplay, like a card being drawn or a bet being made, information is exchanged between the PEA chains and the client applications. The clients publish data about their understanding of the game; this is safe to make public because it is an explicit phase of the Golle algorithm where the players all reach consensus that "the same thing" happened. Once this data is recorded on the PEA chains, it is no longer possible for any peer (including us) to spoof or deny what occurred. Immediately after a client discloses their consensus data, we provide the notary stamp of it back to the client. That means we commit in real-time to clients to the integrity of data just published. That gives everyone the independent capability to verify all parties were acting correctly and above-board, including us. At the end of game when bet accounting is resolved, notary stamps are placed on the Live chain, which enlarges our irrevocable commitment to everyone, not just the players involved in one game

HOW DO THE PEA CHAINS HAVE SUCH HIGH PERFORMANCE?

Two other parameters we can set on PEA chains are the difficulty to find a hash, and the consensus algorithm to accept a new block. Since Live servers are competing with one another, making the hash difficult to find is part of the design. Each server trying thousands of hashes multiplied by thousands of servers is why blockchains

are so inefficient. Then once a block is found, it needs to propagate (potentially competing against other found blocks) until consensus is reached what block actually won. In our PEA system, the servers can cooperate. The server getting a transaction request makes a block for it and transmits it immediately; consensus is also dramatically simplified down to just avoiding collisions, rather than competing for majority.

IF THE PEA CHAINS ARE COPIES OF ETHEREUM, WHY DON'T THEY COST ETH GAS?

Since we can set all the parameters on the PEA chains, we make their gas cost 0. Our chains use DRAW and not ETH, and we absorb the cost of running them in order to provide fast service to game players. Note that there still will be ETH gas costs. When a player begins playing, they must escrow their DRAW into smart contracts on our chains, which costs ETH gas. And when a player is done playing, payouts also cost ETH gas to settle your new account total in DRAW. We can only provide free gas for the duration of game play; all financial accounting done on the Live Ethereum chain still costs ETH gas.

IF PEA IS A FAST-DISTRIBUTED DATA STORE, WHY NOT USE A DIRECTED ACYCLIC GRAPH (DAG)?

DAG-based technologies, such as Hashgraph are very attractive. They offer very competitive performance and efficiency, with similar verifiability and integrity. From a pure technology perspective, DAGs are a sound choice. They are also new and largely untested in practical industry. Blockchains are a more mature technology, for which development tools already exist, with a proven track record in the wild. They also have more mindshare: more people have heard of blockchains

than have heard of DAGs. And since no small part of our viability depends on people feeling they can trust the technologies we're using, we've opted for being more conservative. This also leads us nicely to the next question which is surreptitiously related to this one...

HOW DOES LEAP COMPARE TO OTHER STATE CHANNEL ALTERNATIVES?

Some of our competitors have really solid technology solutions comparable to LEAP. Everyone has seen the need for faster performance and reduced cost, while still retaining assured payments and verifiability, and created different solutions to meet specific needs. The main way LEAP differs from the state channel pack is our ability to service more communities than gamers alone. Since we can cleanly separate public versus confidential information, we can offer greater security and protection to businesses and organizations. Since we designed chain of custody (no pun intended) and impeccable provenance right into the system, we can meet the unique needs of legislators and regulators. And our ability to locally process smart contracts is a major advantage several other state channels lack. There are specific areas where some of our competitors can argue they are technically faster or scale better than LEAP; but the reality is we have traded a few nanoseconds of speed (no one will ever notice) for the capability to offer greater features to a much larger audience.

WHY USE ETHEREUM INSTEAD OF A DIFFERENT BLOCKCHAIN?

Because we've opted for being boldly ambitious. We believe that the public exposure of smart contracts on Ethereum poses an intractable security flaw. Hundred of millions of dollars have already been hacked from Ethereum on multiple occasions. However, despite this known flaw, there is a large and

continuing investment in smart contracts. We are positioning LEAP to be the "better Ethereum" once the current incarnation of Ethereum fails. By the time that happens, we'll have a stable and provably secure alternative for people to migrate their smart contracts to LEAP. To do the same thing they were doing before, just faster, at less cost, and more securely.

WHY IS THE EXECUTION CHAIN KEPT CONFIDENTIAL?

No small part of the security around the Execution chain is to compensate for the known weaknesses of public Ethereum smart contracts (see previous question). But there are other compelling reasons as well. Businesses and large organizations often have a legitimate need for segmented security: to control who sees what information at what time. That means we can offer a variety of LEAP solutions to the business sector, to service what is currently an untapped market. This diversification of revenue streams increases chances of success for us as a business and reduces risk for Draw token holders as an investment.

WHY IS THE AUDIT CHAIN KEPT CONFIDENTIAL?

In order to comply with certain regulations like Anti-Money Laundering, we are required to collect data on user activity that could be used to potentially identify them and their physical location. For most users this data is considered harmless; it's what our browsers transmit with every http request when we surf the web, so we routinely give such data away freely. But some users consider this information private and desire it be kept that way. The Audit chain is confidential to meet the need for privacy from those users. Moreover, we won't disclose user information until legally compelled to do so.

SOME OF YOUR SYSTEM ISN'T OPEN SOURCE. HOW CAN WE TRUST THOSE PARTS OF THE IMPLEMENTATION?

Both extremes of proprietary code and open source have advantages and disadvantages. Open source is typically used to provide assurance but carries the drawbacks of reducing security and exposing confidential information. What most people really want from a system is provable verifiability: an objective guarantee that it does what it says it does. Making software open source is just one of many ways to achieve provable verifiability, and there are better methods available. We have created a new method for black box verification that enables us to show a program meets stated design criteria, without actually revealing the lines of code. This is done using a validator (whose code is entirely open source) with programing logic based on automata theory. This validator has many use cases, for example, it can also be used to assure that code in different languages works the same as a reference implementation. It can also be used on smart contracts, which is how we'll provide verifiability for the Execution and Audit chains. And with some legwork, it can also be used on certain classes of legal contracts, in particular the kinds that interest the B2B community. Taken altogether, this means Blockdraw can do something no one else can currently do: provide the same level of assurance as open source, while keeping business practices and intellectual property protected and secure.

HOW WELL DOES THE LEAP SYSTEM SCALE?

That depends on which part we are talking about. The L is the Live Ethereum blockchain, which scales quite badly. Current blockchain technologies are notoriously inefficient; one estimate finds that more energy in wasted in mining that the entire country of Denmark uses. There are several proposals to make Ethereum more efficient, from moving to a Proof-of-Stake model to the adoption of ethash. On the PEA chains that we've designed, we get good scalability right out of the box. We also have ideas to rapidly evolve the entire system and do things blockchains currently can't. The basic assumption that the PEA servers are allowed to trustlessly cooperate instead of being forced to compete means we have fundamentally better design options available for scaling the entire system.

HOW CAN WE TRUST YOUR EXECUTION IF YOU DON'T PUBLICALLY REVEAL YOUR SMART CONTRACTS?

Publically revealing smart contracts enables open certification; anyone can look at the code and verify it works properly. It turns out that we don't need open certification to ensure trustless cooperation, we only need individual verifiability. Being trustless means that if any participant tries to take advantage of another by cheating, they can always be caught by another participant. So, the answer is, you don't need to trust us; we'll give you all the open source tools needed to verify for yourself that everything we do, and every player does, is above board. Moreover, there's a mathematical proof called the finite state equivalence theorem[1] where if our code meets certain criteria (strongly connected et al) we can show it always plays fairly, without needing to reveal the code. We plan on building a tool whereby anyone can certify code meets the proof requirements without ever seeing actual code. Thus, two businesses could interact with one another fairly and securely, each

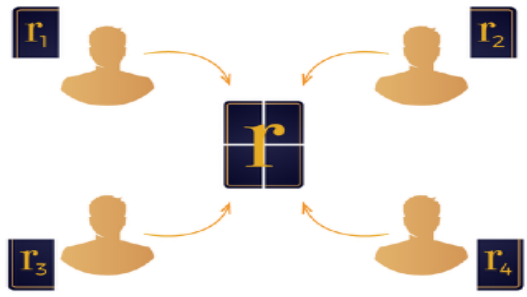
[1] https://en.wikipedia.org/wiki/Finite-state_machine

assured the other is going to honor their contracts, without either needing to know the details of the other's implementation. Our goal is selective confidentiality, not secrecy. Our accounting smart contracts are on the Live chain, and all the calls on the Performance chain are publically viewable, so we disclose quite a bit already. The only thing kept confidential is the precise logic used to run the game or business transaction, and that's both modular best practices in programming and the way companies prefer to operate.

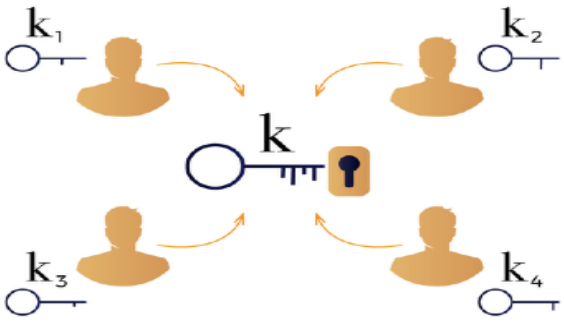
IF YOU HAVE LIVE SMART CONTRACTS (FOR FUNDS ESCROW), WHY DOES LEAP HAVE LESS VULNERABILITY THAN STATE CHANNELS OR OTHER ON-CHAIN GAME SOLUTIONS?

Code vulnerability is a function of several factors: number of lines in the codebase, complexity of its function, how often it is changed, and how many different programmers work on it. The safest kind of code is small, simple, unchanging, and written by one expert and well commented over time (escrow wallets are starting to fall into this category). This is the smart contract (escrow wallet) that we have on the Live chain; it's just an accounting stub that disburses player funds in Draw Tokens. On the other hand, the logic needed to run a game is large, complicated, often updated, and usually has been touched by programmers of varying skill levels. Most state channel implementations place a significant portion of this game logic on the Live chain, which exposes them to risk. We place all that logic on the Execution chain, which is much better protected. We even have a layer of access control from the APIs on the Performance chain, so the Execution code can't even be called except from a secure context. The most dangerous hacks are targeted at specific lines of code that were written by an inexperienced developer, buried within a complex nest of code. In order for hackers to capitalize on subtle mistakes, they need to physically see the lines of code, as is possible on the live chain. Since we keep the code on the Execution chain confidential, we deny the basic capability of

hackers to find susceptibilities. Their only recourse is to try hacks blindly, which is not only less effective, but the set of blind hacks are well known and readily protected against. In fact, most blind hacks aren't even at the code implementation level; these hacks usually involve physical hardware, or are at the network abstraction layer.



Each player chooses a random number $\mathbf{r}_i$. The group adds them all together securely to determine which random card **r** is dealt. No one can manipulate the outcome.



Each player has one piece $\mathbf{k}_i$ of a private key **k**. To reveal a card, all players must cooperate together. No one can see a card they aren't supposed to see, before the rules say they can see it.

IF A JOINTLY HELD PRIVATE KEY IS NECESSARY TO REVEAL CARDS, HOW ARE PLAYERS TIMING OUT OR LEAVING CARD GAMES HANDLED?

Once any part of the jointly held private key is lost, it is no longer possible to decrypt the original deck encryption. At first glance it seems like that might make it impossible to continue the game, but some clever cryptography can salvage the situation. First, we create a new valid deck encryption among all remaining players with a different joint key. Second, each player can claim already revealed cards in their hands (that other players can't see) by furnishing proof-of-work from the original deck encryption applied to the new deck encryption, all without revealing which card they have. The only player who can't do this is the one who dropped,

but that's okay, since no one sees the cards of a forfeit player anyway. Third, the Croupier furnishes the same proof-of-work for any card all players see, like a face up card on a table; this is purely a convenience, since every player could independently provide the same proof. Lastly, all remaining cards are unknown to all players, so are re-dealt from the newly encrypted deck. That means everyone retains all the cards they had before the player dropped, no one knows cards they shouldn't, and all unknown cards still remain unknown. Play then resumes as normal.

DOCUMENT REVISION HISTORY

EACH NEW BLOCKDRAW WHITEPAPER OR TECHNICAL FAQS (THE "WHITEPAPERS") REVISION CONTAINS THE ENTIRE CURRENT UNDER-STANDINGS WITH RESPECT TO THE MATTERS COVERED BY THE WHITEPAPERS. All prior or contemporaneous writings in any of the whitepapers with respect to the subject matter covered hereof and all other such commitments, agreements and writings shall have no further force or effect. Blockdraw is under no obligation to inform readers of any changes to its whitepapers and urges readers to check for revisions from time to time.

Revision	Author	Date	Status and Description
1.0	Administrator	Feb 6, 2018	Initial Draft Techpaper
1.1	Administrator	March 10, 2018	Added Question on Open Source