

BlockQuick: Super-Light Client Protocol for Blockchain Validation on Constrained Devices

Dominic Letz

Exosite

May 24, 2019. Version 0.1

Abstract

Today server authentication is largely handled through PKI infrastructure that is prevalent both in the private as well as the public sector. Most notably established as the defacto standard for internet communication through the world wide web and its usage in HTTPS, SSL/TLS (Web PKI). However, in its application to IoT devices, using the Web PKI infrastructure for server authentication has several shortcomings in validity periods, identity, revocation practice, and governance. Recently different approaches to decentralized PKI (DPKI) using Blockchain technology have been proposed, but so far lacked practicality on constrained devices commonly used in IoT deployments. Existing light client protocols have not been resource efficient enough to be practical. We present BlockQuick, a novel protocol for a super-light client, which features reading blockchain data securely from a remote client, requiring less data than PoPoW or FlyClient for validation while providing effective means to protect against eclipse and MITM attacks on the network. The proposed clients has low kb RAM requirements optimal for IoT devices and embedded MCUs.

Introduction

Web PKI is the defacto standard for encrypted internet communication. Already most internet traffic is being encrypted [\[mad18\]](#) using TLS which relies on PKI for server authentication. This authentication uses X.509 certificates signed by third parties, the certificate authorities (CAs). Since Netscape [\[res01\]](#) brought SSL to the internet with their first browser in 1995, the certification of servers is in the hands of third parties. Over time, many improvements have been made to the Web PKI, still, the following issues around governance, revocation, and handling of time in certificates persist [\[sch10\]](#) [\[hou16\]](#):

Time. For the IoT, the current recommendations of the CA/Browser Forum to shorten certificate lifetimes [\[hou16b\]](#) are especially heavyweight. In contrast to personal computers,

many IoT devices have much more limited connectivity and power supply. In many use cases, in fact, the devices can be offline e.g. in storage or on the shelf for many years. When an IoT device connects the first time to a network, it thus needs to establish two things: 1) The actual current time 2) Secure validated connectivity. This poses a chicken or egg problem because without the actual current time the device can't validate Web PKI certificates, and without Web PKI certificate validation it does not have a trusted source for time.

Previous solutions to this problem often trade-off security in order to work-around this problem. Such as:

- Falling back to read time from non-authenticated time sources [\[alr18\]](#) such as NTP [\[wic18\]](#), GPS [\[kar17\]](#)
- Accept insecure time from the same server offering the certificate to be validated [\[tsc15\]](#)
- Usage of a hard-coded factory build time timestamp when the real-time is not known

This problem is further amplified with ever shorter lived certificates, as that means that certificates stored on the device, such as the root certificates or cached endpoint certificates might all have expired. This is more so an issue as many of today's resource-efficient IoT devices store very little data - often less than the recommended minimum of Web PKI root certificates. Without a trustworthy understanding of time and potentially expired certificates IoT devices are prone to man-in-the-middle attacks [\[sel15\]](#) and fake time servers [\[mal15\]](#).

Multiple Certificates. Next to the issue of time PKI has by design no method to detect duplicate identities. It is not possible for any peer of the system to know how many certificates represent the same identity. This enables attacks using alternative sets of certificates without the user, the victim or the certificate authority knowing. [\[gre17\]](#)

Governance. This design problem in connection with the governance structure of Web PKI creates a third issue. In Web PKI there are at the time of writing 3,625 valid intermediate certificates [\[cen19\]](#). Each of these certificates can be used by their holders to create valid certificates for any domain. Unfortunately, misuse is [\[kim17\]](#) and has been common [\[ven14\]](#) [\[lav14\]](#).

Revocation. Lastly, revocation is a necessary part of the certificate lifecycle and defined in PKI via CRL and OCSP. Unfortunately, implementations vary especially on IoT devices are usually missing and adoption from service providers to actually announce revoked certificates via these mechanisms is very low [\[liu15\]](#). For any actual IoT use case, one has to summarise that Web PKI revocation is unfortunately non-functional.

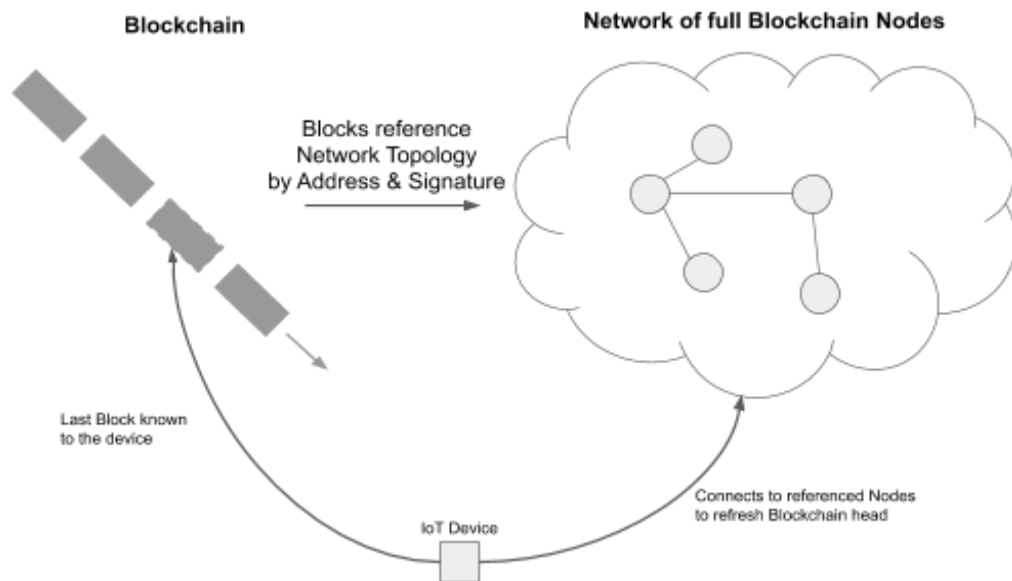
DPKI. In response to these shortcomings, decentralized public key infrastructure based on blockchain technology has been proposed [\[but15\]](#) [\[sin17\]](#) [\[fro14\]](#) [\[pat18\]](#). Deploying a public key infrastructure on a decentralized blockchain network has the benefits of alleviating all four mentioned PKI shortcomings: time, multiple certificates, governance, revocation. Though authors agree on these theoretical benefits adoption of DPKI among

IoT devices so far has been hindered by the absence of practical light client implementations [\[pac32\]](#) [\[mag18\]](#). In this paper we want to introduce a new light client protocol that can be added to the Ethereum [\[but14\]](#) blockchain, allowing DPKI constructions to use Ethereum smart contracts without locking out constrained devices.

Related light clients. In recent work Bünz et.al. have proposed with FlyClient [\[bun19\]](#) the super-light client class of light clients for use in the Ethereum network. The super-light client needs to download only a logarithmic amount of block headers for validation but requires the availability of at least one honest miner. In the context of IoT devices, this restriction seems unrealistic to us as many remote devices are easy targets for network manipulation such as eclipse attacks [\[hei15\]](#). Individual devices can often be isolated when attackers intermediate cellular networks [\[sha15\]](#) [\[mey04\]](#) [\[rij15\]](#) or wifi networks [\[van18\]](#) and run Man-In-The-Middle (MITM) attacks there [\[lee19\]](#). Also, this is true for any mobile devices that can be stolen and placed into a maliciously constructed network, with no honest miners at all. As such a secure light client must have the ability to identify when no honest miner is available at all, e.g. when the clients' local network has been manipulated.

Contribution: BlockQuick

We introduce BlockQuick, a new super-light client protocol for the Ethereum, and similar blockchains that has sublinear in fact near constant bandwidth requirements for chain validation while being resistant to eclipse and MITM attacks. It enables a client to sync up to a relatively recent block (dependent on a chosen parameter Δt) using a **consensus reputation table**. The amount of data that needs to be synced from blockchain nodes is independent of the block height, or the total number of existing blocks, but depends on the historic majority change rate of miners within the consensus reputation table. We show that this means for blockchains such as Bitcoin and Ethereum a near constant amount of data that needs to be synced - Below 50kb for Ethereum and ~20kb for a further modified variant thereof.



Blockchain Additions:

The BlockQuick protocol can be added to existing blockchains but requires each mined block header to contain the network peer information of the block miner - internet address and public id - and a corresponding cryptographic signature of the header. In case of recoverable ECDSA signatures, as used in Bitcoin and Ethereum, the public key is already part of the signature and hence does not need to be an explicit part of the block header.

The additional required consensus rules for full nodes on the network are:

- Each block needs to have a cryptographic signature (and public key if not included in the signature).
- Each block needs to contain a Merkle tree of all previous block headers.
- Each miner should be reachable under the specified internet address.

BlockQuick Client Protocol

A client needs at a minimum a single last known blockchain block header hash. In an IoT use case, this might be at the start a recent block header hash at the time of manufacturing. Additionally, the client needs either a list of seed nodes or a mechanism such as DNS to look seed nodes up. These seed nodes do not correspond to a list of trusted nodes - we will see how the protocol is able to trust data without needing to trust the nodes supplying the data.

Constructing the **Consensus Reputation Table**. The consensus reputation table is not centrally provided to clients but is constructed one each client based on the last known block header. For example, in a Proof-of-Work (PoW) system the client can construct the consensus reputation table by reading 100 previous block headers from the last known state

of the blockchain. All past block headers of the blockchain can be validated on the client using the parent block checksum alone. Thus clients can even fetch these block headers from untrusted nodes as the data can be validated using the existing block header and its contained parent block hash. Once the client has those past block headers fetched and validated, it creates the reputation table from the information in the headers as follows:

The proposed light client protocol establishes a reputation system on the mining nodes in the network. The reputation of each mining node corresponds directly to the percentage of blocks that each mining node contributed compared to all mining nodes in a given time frame. The time frame is the consensus groups history length parameter Δt in the number of blocks. In a PoW consensus, the percentage of mined blocks during the last Δt blocks corresponds to their proportional computational proof power among all participants. A miner who mined and signed 10% of the Δt recent blocks thereby shows to have 10% of the total computational power during that time frame, generalized for other consensus algorithms we call this their **consensus share**.

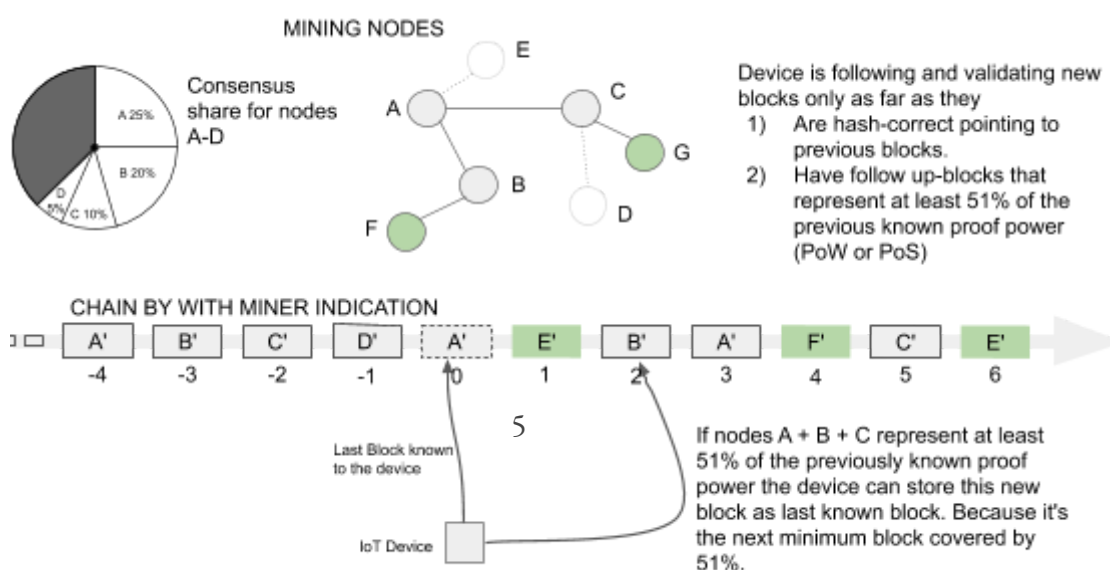
As such clients can cache a list of most reputable miners together with the most recent known good block for a quicker update mechanism. Each miner thereby is stored as a triplet of the peer's public key, its address, and its consensus share. The client needs to store at least N reputable miners so that the sum of their consensus share is $> 50\%$ but should store more up to Δt miners depending on local storage allowance.

Simple Connection Algorithm

If the client has an existing cached list of last miners it tries to reach enough miners such that the sum of their consensus share is at least $> 50\%$ in the Quick Update process:

Quick Update - For each miner M in the client's miner list:

- (1) Client connects to the miner and fails if not found or not matching the stored public key
- (2) Client downloads most recent block headers from the miner.
- (3) If the client successfully reached enough miners corresponding to $> 50\%$ consensus share the client can stop



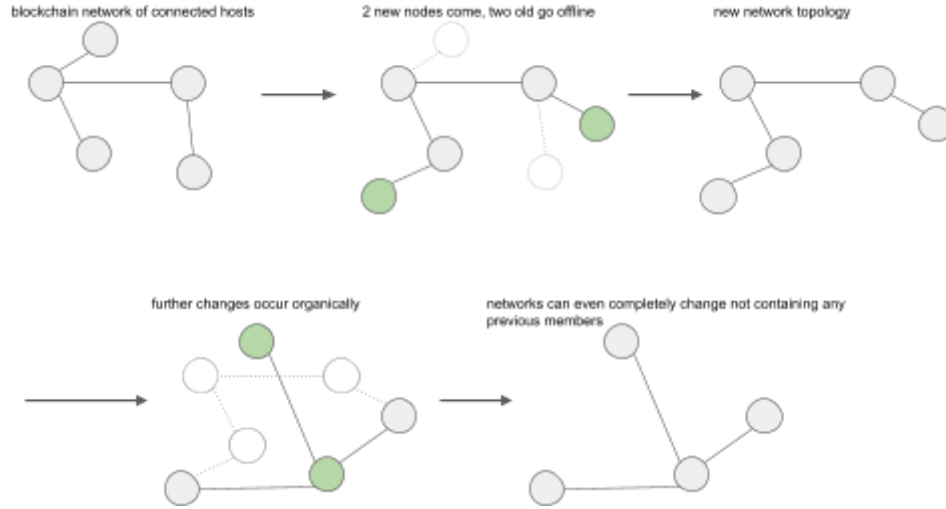
Slow Update - If less than 50% of the consensus share could be connected to the device assumes that a "long period of time" has passed and falls back to this slower update sub-routine:

- (1) The client defines the current consensus group as the Δt last blocks from its last known blocks. Based on this consensus group the consensus share of each peer in that consensus group is calculated. E.g. if $\Delta t = 100$ and miner A created 25 of the last 100 blocks his consensus share is 25%.
The data is either cached on the client or can be downloaded from any peer, validating the block-header data based on the existing block header hash on the client.
- (2) Then the client downloads the most recent block headers from any peer
- (3) The client further fetches incrementally the whole missing link of block-headers between last known block-header.
- (4) For each new block header, the client assigns a score based on the known consensus share of the block miner or zero if the block miner is not known.
For example in the above picture, block 1 mined by miner E receives a score of 0, since its corresponding miner E is not part of the previous consensus group.
- (5) The score value of each new block is then increased once per miner from the known consensus group having a block in the following block list. The logic is that when a miner B created and signed a block on this candidate branch of the blockchain, it means the miner must trust this branch.
So when the client sees block 2 mined by miner B it receives a positive score of 25% and the score of block 1 is also increased to 25% - because it has now been confirmed by a miner who represents 25% of the consensus group.
- (6) In case no block reaches the threshold score of >50% the device will disconnect and assume manipulation.
- (7) With each found block above the 50% threshold, the consensus group is updated to now include that block, allowing the consensus shares to adjust over time, mirroring the shift in the consensus group in the real world when miners shut down their operation or new miners start operation.
- (8) Finally, when the >50% threshold has been established for some new blocks but cannot be established for further blocks (e.g. because there are not enough further blocks) the client repeats the Quick Update process based on the new current consensus group. Only if the quick update process succeeds, this the most recent block above the 50% threshold is now stored as the new most recently known block on the device.

During the execution of this algorithm, the client might encounter multiple different versions of the blockchain, potentially malicious forks and outdated versions. But unless an attacker manages to gain more than 50% of the consensus share by their private keys, the client will not accept a wrong chain. For outdated but valid blocks though it is important

that the client always executes the Quick Update after a Slow Update to validate with multiple miners, that this is still the most recent version of the blockchain.

The present algorithm allows a device to follow the blockchain through gradual changes of

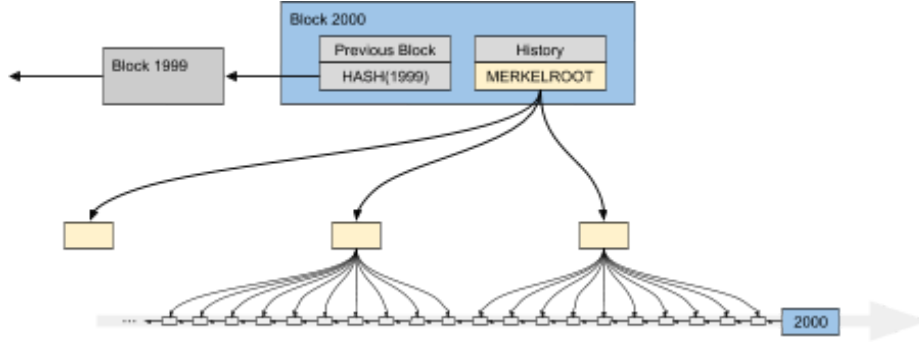


the consensus group as long as the impact of the change within the Δt last blocks is below 50% of consensus share.

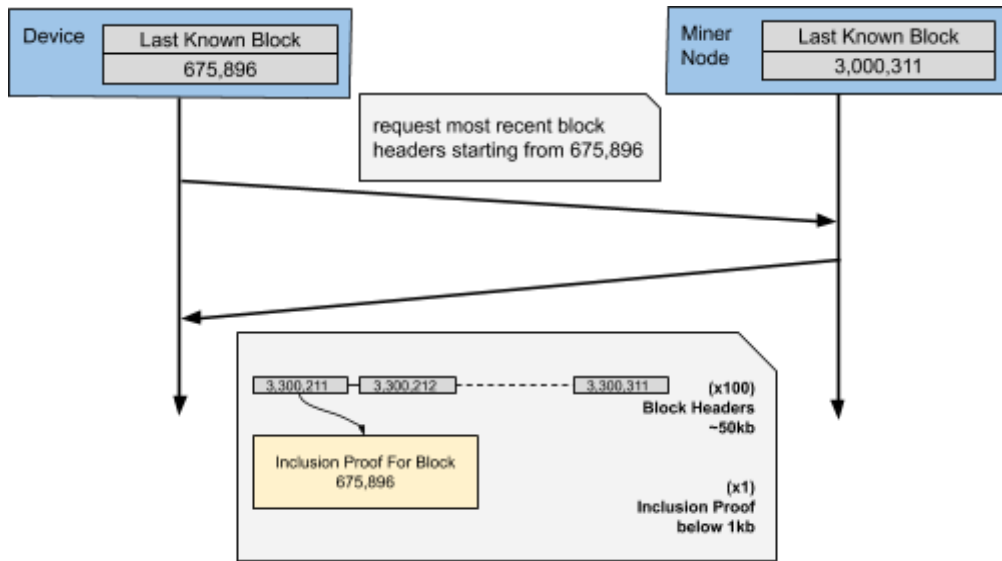
Sublinear Sync Size

The presented solution enables IoT devices to establish trust in a recent blockchain block, by operating on block headers and verifying the miner signatures. In existing blockchains such as Ethereum and Bitcoin, an inclusion proof for previous blocks is given with the previous block hash of each block. Validating this inclusion proof though requires iterating all blocks linearly in this order. Given the current size of the Ethereum headers and for a device that was offline for an extended period the incremental check would take too much time and bandwidth to be realistic. E.g. given 3 years since the last known block and an average block header size of 500 bytes at one header every 15 seconds we're looking at a total of 6,307,200 blocks a total of ~3gb that would need to be downloaded and checked by the IoT devices. This obviously is not feasible.

To address this, there are two adjustments compared to Ethereum. First, merkelized inclusion proofs can be utilized as described by Crosby and Wallach [\[cro09\]](#) and Bünzel et.al. later [\[bun19\]](#). With this construction in a new "History" field in each block, a client can validate the inclusion of any previous block based on a Merkle proof. The Merkle inclusion proof has only a logarithmic size and sending all headers in between the devices last known block and the current most recent block can then be skipped.



Secondly, when a device connects to a miner, not only does it request the most recent block header but it also provides its last known block header and Δt . This information allows the server/miner to determine whether there were majority changes in the consensus group



between the last known state on the client and current state of the blockchain. The server/miner can then calculate the minimum number of block headers that the device needs to synchronize.

Using this approach, the number of blocks that needs to be fetched even after a long period is majorly reduced. From any point in the blockchain known to the device, the server/miner node only needs to provide a Merkle proof to a recent block, plus the N following blocks required for the device to confirm these blocks based on its known, consensus group. So, in the example of bridging three years or 6,307,200 blocks, the device would fetch a Merkle inclusion proof of the logarithmic size of the total blockchain. We estimate an upper worst case using the [\[cro09\]](#) binary Merkle tree and 50 years or 105 million blocks. The Merkle proof would thus require $\log_2(105,000,000) = \sim 26$ nodes @ 32 byte each for SHA-256, totaling 832 bytes for the inclusion proof. Additionally, the most recent Δt block headers are required to build the new consensus group and validate the approval of the current consensus group. At 500 bytes per header and $\Delta t = 100$, those headers accumulate to 50kb, totaling the required data transfer at ~ 51 kb.

Majority Change. Over the lifetime of a blockchain, the consensus group will have incremental changes. New miners come online and old miners stop their participation in the network. Over an extended period of time, this means that the consensus group that has been known to the device and the consensus group of the current longest chain might significantly differ. At some point, the difference might be so large that the last known consensus group won't be sufficient to confirm the recent blocks with $>50\%$ confidence, as there are not enough known signatures present anymore after a certain point in the chain. In this case, the device would not be able to accept the most recent Δt blocks directly as they can't be confirmed by its last known consensus group. In this case, the server needs to provide additional blocks around the time of the consensus group change to allow the client to reconstruct the gradual change of the consensus group. Effectively, this provides a first incremental sync to a new blockchain state that is still confirmable by the devices last known consensus group. If necessary there can be an arbitrary amount of these incremental steps.

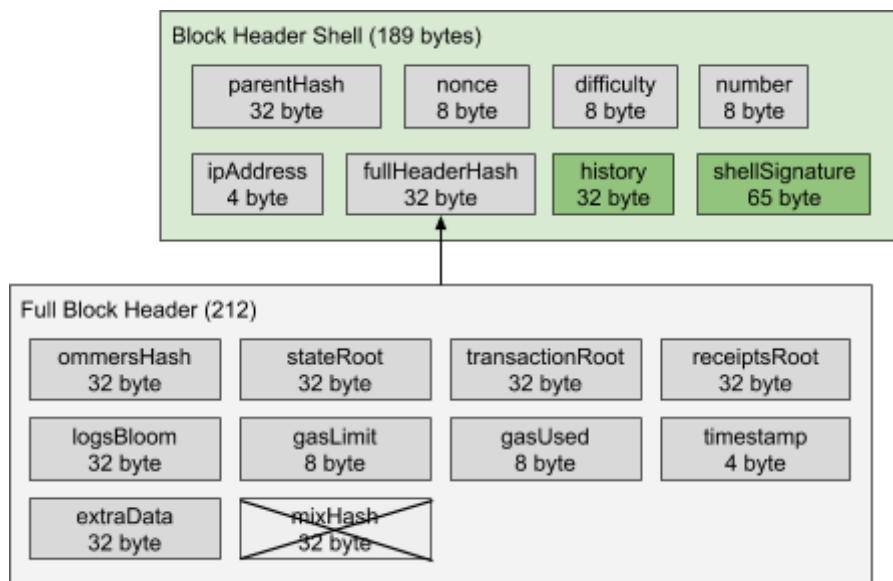
Given a blockchain state A known to the device and a consensus group G_a at that point, then there can be a blockchain state B behind which $>50\%$ of the members of G_a by consensus share are not participating in the network anymore - not producing and signing block anymore. If such a blockchain state B exists then the miner/server needs to provide the device the blockchain state immediately before B that was still signed by more than 50% of the devices known consensus group by consensus share. It is this amount of $>50\%$ consensus group changes that determine the total size of data required for the proof. For each majority change of the consensus group, there is at least one inclusion proof to that block, as well as Δt blocks around this change necessary for the client to update its consensus group.

In a highly volatile blockchain, this could degrade to be linear to blockchain length, meaning that every Δt blocks a majority of the consensus group by consensus share has changed. However, when looking at all past blocks of Ethereum as a reference we can determine that while major changes in the consensus group were frequent within the first at the beginning of the blockchain with growing adoption the change of the consensus group has slowed down. During the last 5 million blocks there was no major consensus group change anymore, which means that a device would not need to sync any intermediate states.

Block Header Shelling

As discussed above, the total needed data transfer size using the BlockQuick algorithm linearly depends on the three factors of 1) block header size 2) consensus group history length Δt and 3) consensus change rate. While the consensus change rate is dependent on the economy of the blockchain and consensus protocol, the consensus group history length is defined by the client, based on security requirements.

The block header size though is depending on the chain implementation. In the case of current Ethereum [\[woo19\]](#) the header size is around 500 bytes, but it can be optimized by partitioning the full header into two parts: 1) An outer block header shell and 2) an inner full block header. The outer block header shell is reduced to only include the information required to validate using the BlockQuick algorithm: parentHash, nonce, difficulty, number, address, history, signature as well as a 256bit hash of the inner full block header. The second partition - the full block header does not need to be transferred to the client during validation up until the client was able to confirm a block. Only of the most recent confirmed block the client will need the full block header to access stateRoot, transactionRoot et cetera to perform further application-specific functions.



In the case of PoW, as pictured above, the relevant signature and difficulty are captured in the shell. This header partitioning applied on Ethereum would further cut the maximum required data by more than half. Using the same exemplary calculation as before but now with block header size = 189, consensus group history length $\Delta t = 100$, and no consensus majority change, we estimate 20kb of total sync size on average.

Summary

Empirical studies [lec11] have shown that misused private keys and stolen certificates are the most common risks in the PKI. We propose a new secure blockchain client that makes it feasible to implement DPKI solutions, replacing or complementing [bal17] Web PKI through eliminating the risks introduced by third parties. By adopting the proposed approach, IoT devices are enabled to securely read blockchain information including the current timestamp and registered DPKI information about domains, certificates, and ownership. Further, the Ethereum Merkle roots allow the clients to validate each of these pieces of information against the last known good block.

We've shown a novel method to allow constrained devices typical to the IoT space to consume blockchain based data. This is the first building block to enable a more secure DPKI with blockchain based smart contracts. Devices are enabled to fetch block time and arbitrary state data from the Blockchain. In comparison to existing clients for the Ethereum network, we can understand the magnitude of the impact.

Node Type	Trust Model	Storage	Bandwidth	Duration
geth --syncmode=full	Decentralized	~220GB ¹	>90GB ²	days
geth --syncmode=fast	Decentralized	~130GB ³	>90GB ²	hours
geth --syncmode=light	Decentralized	~1.2GB ³	~1.2GB ³	minutes
BlockQuick	Decentralized	~20kb ³	~20kb ³	sub-second
Trad. Web PKI Client	Centralized	~20kb - 500kb ⁴	~5kb ³	sub-second

At 20kb or below our solution is still above the handshake size of a traditional centralized Web PKI Client, but it becomes feasible to execute on most hardware that is capable of Web PKI handshakes.

¹ Retrieved from <https://etherscan.io/chartsync/chaindefault> on May 10th 2019

² Estimated based on total block sizes <https://etherscan.io/chart/blocksize> on May 15th 2019

³ As measured on May 10th 2019

⁴ Range from minimal four AWS IoT certificates (20kb) and max all Mozilla NSS certificates (500kb) as of May 10th 2019

<https://docs.aws.amazon.com/iot/latest/developerguide/managing-device-certs.html>
<https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/>

Potential Attacks

Chain Forks and Network Splits. In Ethereum, Bitcoin and other existing PoW networks a fork chain can be created by a few malicious miners or even just a software bug. For whatever reasons such as fork occurs the question, whether a client would still be able to connect to the blockchain, and whether it would be able to detect one of the forks as authoritative, while the other is in fact a fork that should be ignored. From the perspective of a client device using the BlockQuick algorithm the deciding factor is always the consensus share that each block can achieve.

It is for this reason that the threshold for accepting a block in BlockQuick is $>50\%$ of the known consensus share. This ensures that from the device perspective during a network split there is either no authoritative network or just one. By example, say there is a large network split e.g. due to atlantic network cables failing, in this case we have two new networks A and B. Each active miner and client sees either network A or network B depending on their geographic location. Given a consensus reputation table from just before the fork we know that the consensus share S of the sum of all miners from A and all miners from B must have been 100%

$$\sum_{m \in A} S_m + \sum_{m \in B} S_m = 1$$

From there it follows that there can be no split between A and B such that both sides share is bigger than 50%. We can differentiate three cases:

$$\alpha : \sum_{m \in A} S_m > 0.5 \rightarrow \sum_{m \in B} S_m < 0.5$$

Network A is authoritative with more 50% consensus share, clients will accept it's blocks.. Network B is a fork, and no block from B will be accepted by a client.

$$\beta : \sum_{m \in A} S_m < 0.5 \rightarrow \sum_{m \in B} S_m > 0.5$$

Network B is authoritative with more 50% consensus share, clients will accept it's blocks.. Network A is a fork, and no block from B will be accepted by a client.

$$\gamma : \sum_{m \in A} S_m = 0.5 \rightarrow \sum_{m \in B} S_m = 0.5$$

Neither network A nor network B can reach more than 50% consensus share and thus no network is authoritative.

While case γ is unlikely already, e.g. exactly 50% of the miners would need to be both sides of a split, the scenario of exactly 50% consensus share on both sides can be made at mathematically impossible by choosing an odd number for Δt . E.g. choosing $\Delta t = 101$

would not allow a 50%/50% split but A=51 vs B=50 or vice versa in the worst case, making only one network the authoritative survivor.

Eclipse Attacks. Eclipse attacks as discussed by Heilman et.al. [\[hei15\]](#) were directed at node-to-node communication within full-nodes of the Bitcoin peer to peer network. For remote clients as in super-light client use cases network uplinks are typically even more scarce. A mobile IoT device will usually have a single uplink to the internet such as a cellular network or use wifi networks in its vicinity. In these cases eclipsing all connections that the device has to the outside world becomes much easier. Lee et.al [\[lee19\]](#) showed how in a wifi network a device can be spoofed using crafted ARP packets, routing all data traffic through the attacker. This way of monopolizing all connections of the device allows for advanced manipulation, such as presenting the device with multiple different non-authoritative blockchain variants. PoW light clients such as Ethereum geth that validate based on longest chain / highest difficulty are prone to these attacks because they can only successfully choose "the best" chain given a set of alternatives. But when all presented alternative chain versions are non-authoritative the client will still accept one of them. The same is true for FlyClient as proposed by Bünz et.al. [\[bun19\]](#). Here a heuristic comparison of presented alternative chains is done. In the result the FlyClient can decide which of these alternatives is most likely the correct chain. These clients can successfully solve the decision problem on which of presented alternatives is most likely the authoritative chain, but they fail to identify an abort criteria, when no presented chain versions is authoritative.

BlockQuick addresses Eclipse Attacks and specifically the extreme when a single actor monopolizes all connections using the above 50% consensus threshold. Let's assume an attacker constructs a forked blockchain with 10 blocks of wrong data entries. The authoritative version of the PoW chain would in this case grow quickly longer than this forked wrong chain, but devices without a direct connection to the authoritative longer PoW chain would not be able to see the difference. In BlockQuick however, the PoW difficulty and length are not the primary deciding factors. Instead devices will iterate the blocks, in this case 10 new blocks that the device is not aware of and run the Slow Update mechanism as described above. In each new block the device validates the cryptographic signature of the miner and compares these with the known miner identities in its **consensus reputation table**. Only if a block is receiving a consensus score of >50% it is accepted by the client. Receiving a chain with a lower total score results in an abort. Since all 10 new blocks from the example have been crafted and signed by the attacker they on their own would not receive any reputation score and not cross the >50% threshold required for the client to accept these new blocks. Furthermore, since this is a forked chain there are no other miners who would contribute on-top of this fork and thus there are also no other later signatures that would increase the reputation score. Without additional signatures from existing reputable miners in the blockchain, it does not matter how much longer the attackers fork chain becomes 10s, 100s or 1000s of blocks, their reputation score will stay 0% and the

BlockQuick client won't accept them. As a result, the attacked client device would reject the offered blockchain as non-authoritative and close the connection.

Discussion

Blockchain Time Source. As Adam Langley pointed out [\[lan14\]](#) today secure time mechanisms have a circular dependency between knowing the time and having the ability to interpret secure certificate validity dates. With BlockQuick there is now - for the first time we believe - a protocol to break this circular dependency. In our approach the trust anchor that is stored on a computing device is a cryptographic hash of the last known good block header. SHA-3-256 / KECCAK-256 in current Ethereum. The benefit of using a cryptographic hash instead of a public/private key pair is that there is no individual entity that can be attacked to gain access to a private key part, and no third party that could go rogue with a private key.

BlockQuick can only be used for a rough understanding of time though, enough to validate certificates but not precise enough for many other tasks. Ethereum for example has a 15 second block interval, this is the minimum latency for our calculation. Clients can though never confirm up to the most recent block - In the worst case $\frac{\Delta t}{2}$ blocks are required past the next confirmed block, which in the case of $\Delta t = 100$ would lead to a mean latency of $(15 \cdot 100 / 2) = 750$ seconds. For clients which need higher accuracy, it still allows breaking the initial circular dependency on checking certificate validities. With this rough understanding of time clients can now connect to a time server using PKI validation to improve the accuracy of the retrieved time. Authoritative accurate time servers could in fact be registered in blockchain data directly using certificate pinning as discussed below.

Certificate Pinning. We have presented the foundation for constrained devices to securely read recent blockchain data. We propose to solve aforementioned PKI issues by complementing Web PKI with pinning certificates in a distributed blockchain such as Ethereum. Pinning a certificate is a means of storing a certificate hash which then can be used to validate the concrete identity of a certificate. This combination solves the aforementioned problems of PKI: Time, Revocation, and Multiple Identities as the blockchain storage becomes the authoritative registry for the currently valid certificate for any entity. For Governance there are alternatives in development such as ENS [\[joh19\]](#) but the description of a complete solution in the form of a smart-contract structure is an ongoing research area.

Too fast consensus share changes. The BlockQuick algorithm does detect wrong blockchain branches based on the consensus group score. When the score does not grow above the threshold of 50%, the algorithm discards the branch. This thought puts a limit on the maximal consensus group change rate. If there exists any block after which the majority of the consensus group has changed (e.g. 51% of all miners suddenly stop mining), the BlockQuick algorithm would not consider the chain authentic. Should there be a

catastrophic event leading to such a sudden loss of the majority, the blockchain would effectively stop. This is a new theoretical limitation directly resulting from the design. Our testing on Ethereum history though showed that neither chain ever had such an event so far. It is future work to closely investigate the likelihood of such an event.

Selection of Δt . Increasing the size of Δt also increases linearly how much data has to be fetched for validation. At the increased size of Δt reduces consensus group fluctuations stored on the devices. Higher fluctuation can lead in the worst case to a perceived too fast consensus share change for the devices to follow. Implementations thus may choose different Δt appropriate. Additionally, the blockchain consensus may be extended to enforce changes rates of less than 50% within a Δt timeframe within their consensus definition. This might also render small Δt values practical. Future work may investigate this further.

References

- [alr18] Alrawi, O. and Lever, C. and Antonakakis, M. and Monroe, F. [SoK: Security Evaluation of Home-Based IoT Deployments](#), 2018
- [bal17] Baldi, M. and Chiaraluce, F. and Frontoni, E. and Gottardi, G. and Sciarroni, D. and Spalazzi, L. [Certificate Validation through Public Ledgers and Blockchains](#), 2017
- [bun19] Bünz, B. and Kiffer, L. and Luu, L. and Zamani, M. [FlyClient: Super-Light Clients for Cryptocurrencies](#), 2019
- [but14] Buterin, V. [Ethereum](#), 2014
- [but15] Allen, C. and Brock, A. and Buterin, V. and Callas, J. and Dorje, D. and Lundkvist, C. and Kravchenko, P. and Nelson, J. and Reed, D. and Sabadello, M. and Slepak, G. and Thorp, N. and Wood, H.T. [Decentralized Public Key Infrastructure](#), 2015
- [cen19] Censys [NSS Certificate Report](#), 2019
- [cro09] Crosby, S.A. and Wallach, D.S. [Efficient Data Structures for Tamper-Evident Logging](#), 2009
- [fro14] Fromknecht, C. and Velicanu, D. and Yakoubov, S. [A Decentralized Public Key Infrastructure with Identity Retention](#), 2014
- [gre17] Greenberg, A. [How Hackers Hijacked a Bank's Entire Online Operation](#), 2017
- [hei15] Heilman, E. and Kendler, A. and Zohar†, A. and Goldberg, S. [Eclipse Attacks on Bitcoin's Peer-to-Peer Network](#), 2015
- [hou16] Housley, R. and O'Donoghue, K. [Problems with the Public Key Infrastructure \(PKI\) for the World Wide Web](#), 2016
- [hou16b] Housley, R. and O'Donoghue, K. [Problems with the Public Key Infrastructure \(PKI\) for the World Wide Web: 3.2.1. Short-lived Certificates](#), 2016
- [joh19] Johnson, N. [ENS Documentation](#), 2019
- [kar17] Dave/Karit [Using GPS Spoofing to Control Time](#), 2017
- [kim17] Kim, D. and Kwon, B.J. and Dumitras, T. [Certified Malware: Measuring Breaches of Trust in the Windows Code-Signing PKI](#), 2017

- [lan14] Langley, A. [Expired HTTPS certificates and incorrect clocks](#), 2014
- [lav14] Delignat-Lavaud, A. and Abadi, M. and Birell, A. and Mironov, I. and Woobler, T. and Xie, Y. [Web PKI: Closing the Gap between Guidelines and Practices](#), 2014
- [lec11] Lechner, A. [Low Probability, High Stakes: A look at PKI](#), 2011
- [lee19] Lee, D. and Kwok, E. and Yee, D. [KRACK - The Destruction of WiFi: A simulation of KRACK attack](#), 2019
- [liu15] Liu, Y. and Tome, W. and Zhang, L. and Choffnes, D. and Levin, D. and Maggs, B. and Mislove, A. and Schulman, A. and Wilson, C. [An End-to-End Measurement of Certificate Revocation in the Web's PKI](#), 2015
- [mad18] Maddison, J. [Encrypted Traffic Reaches A New Threshold](#), 2018
- [mag18] Magnusson, S. [Evaluation of Decentralized Alternatives to PKI for IoT Devices](#), 2018
- [mal15] Malhotra, A. and Cohen, I.E. and Brakke, E. and Goldberg, S. [Attacking the Network Time Protocol](#), 2015
- [mey04] Meyer, U. and Wetzel, S. [A Man-in-the-Middle Attack on UMTS](#), 2004
- [pac32] Unknown [Blockchain takes way too long to sync-issue #2394-ethereum/mist](#), 2017
- [pat18] Patsonakis, C. and Samari, K. and Roussopoulos, M. and Kiayias, A. [Towards a Smart Contract-based, Decentralized, Public-Key Infrastructure](#), 2018
- [res01] Rescorla, E. [SSL and TLS: Designing and Building Secure Systems](#), 2001
- [rij15] Rijdsbergen, K.v. [The effectiveness of a homemade IMSI catcher build with YateBTS and a BladeRF](#), 2015
- [sch10] Schneier, B. [Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure](#), 2010
- [sel15] Selvi, J. [Breaking SSL using time synchronisation attacks](#), 2015
- [sha15] Shaik, A. and Borgoankar, R. and Asokan, N. and Niemi, V. and Seifert, J.P. [Practical attacks against privacy and availability in 4G/LTE mobile communication systems](#), 2015
- [sin17] P, S. and Singh, D.K. [Privacy based decentralized Public Key Infrastructure \(PKI\) implementation using Smart contract in Blockchain](#), 2017
- [tsc15] Tschofenig, H. [\[Dtls-iot\] Secure Time \(again\)](#), 2015
- [van18] Vanhoef, M. and Piessens, F. [Release the Kraken: new KRACKs in the 802.11 Standard](#), 2018
- [ven14] Venafi [Broken Trust: Exposing the Malicious Use of Digital Certificates and Cryptographic Keys](#), 2014
- [wic18] Wicker, G. [Using Device Time to Validate AWS IoT Server Certificates](#), 2018
- [woo19] Wood, G. [Ethereum Yellow Paper](#), 2019