Teechan: Payment Channels Using Trusted Execution Environments

Joshua Lind[1], Ittay Eyal[2], Peter Pietzuch[1], and Emin Gün Sirer[2]

[1] Imperial College London
[2] Cornell University

Abstract. Blockchain protocols are inherently limited in transaction throughput and latency. Recent efforts to address these performance metrics and scale blockchains have focused on off-chain *payment channels*. While such channels can achieve low latency and high throughput, deploying them securely on top of the Bitcoin blockchain has been difficult, partly because building a secure and robust implementation requires changes to the underlying protocol as well as the overall ecosystem.

We present *Teechan*, a full-duplex payment channel framework that exploits *trusted execution environments*. Teechan can be deployed securely on the existing Bitcoin blockchain without having to modify the protocol. Teechan: (i) achieves a higher transaction throughput and lower transaction latency than prior solutions; (ii) enables unlimited full-duplex payments as long as the balance does not exceed the channel's credit; (iii) requires only a single message to be sent per payment in any direction; and (iv) places only two transactions on the blockchain under any execution scenario.

We have built and deployed the Teechan framework using Intel SGX on the Bitcoin network. Our experiments show that Teechan can achieve 2,480 transactions per second on a single channel, with sub-millisecond latency.

1 Introduction

Bitcoin has grown significantly in popularity since its release. The ability to transfer funds over a trustless, decentralized and global financial network has attracted many different industries and applications. As a result, adoption has grown rapidly, leading to an exponential increase in the number of transactions sent per day [4]. This growth exacerbates a natural problem: the consensus protocol that underpins Bitcoin is fundamentally limited in transaction throughput and imposes a minimum transaction latency [11]. Furthermore, since miners must store the history of every transaction ever made, accumulating storage costs increase the cost of running nodes, which, in turn, leads to centralization pressure.

At present, the maximum transaction throughput of Bitcoin is determined by the *block size* and the *block interval*. With a block size of 1 MB and an average block interval of 10 minutes, Bitcoin can support a hard maximum of 7 tx/s [9]. Recent proposals have suggested either tweaking these parameters, such as increasing the block size or reducing the block interval [16,1,15,35]; or modifying the protocol, for instance by incrementally creating the blocks so as to avoid centralization bottlenecks and increase throughput [14]. The former approach cannot scale Bitcoin by more than a single order of magnitude, while the latter requires changes to the underlying protocol that the practitioners have been reticent to make. Other research suggests that hardware limits, such as the cost of signature verification and storage latencies, fundamentally cap Bitcoin to 200 tx/s [12].

To handle demanding workloads, such as credit card processing ($\geq$ 10,000 tx/s), recent proposals have focused on moving transactions off the blockchain through the use of point-to-point *payment channels* [17,12,30]. Payment channels allow for efficient, trustless fund transfers, in which the two parties can exchange transactions without having to modify the blockchain except when the channel is *established* or *terminated*. Consequently, two parties can engage in a large number of fund transfers, only settling the net result on the blockchain. This decreases transaction confirmation latency, as only two entities are involved, and reduces the load on the blockchain and network: throughput scales linearly with the number of channels.

Despite the advantages of payment channels, none have been deployed securely on the existing Bitcoin network, as they require modifications to the underlying Bitcoin protocol. Specifically, they require transaction IDs to be set before they are signed—a proposal to address this issue, SegWit [25], is currently mired in controversy [8].

We present **Teechan**, the first *high-performance micropayment channel framework* that supports practical, secure and efficient fund transfers on the current Bitcoin network. Similar in design to duplex payment channels and the Lightning Network, Teechan uses multi-signature time-locked transactions to establish long-lived payment channels between two mutually distrusting parties. It fundamentally differs from existing protocols, however, in that it leverages *trusted execution environments* (TEEs), as found in recent commodity processors such as Intel CPUs with Software Guard Extensions (SGX). This key enabling technology behind Teechan drastically strengthens the guarantees provided by the framework: (i) Teechan does not require any changes to the existing Bitcoin network, (ii) it enables infinite channel reuse as long as the balance does not exceed the channel limits, and (iii) it is space-efficient, requiring only two transactions to placed in the blockchain in total. Section 6 discusses the comparison to prior art in detail.

At a high level, secure hardware can provide confidentiality and integrity guarantees for code and data, but, by itself, it cannot provide liveness guarantees for a protocol, nor can it ensure safe termination for a network protocol. The Teechan framework is designed in such a manner that, despite these limitations, no party can gain access to more funds than their current net balance. In particular, the secure hardware ensures that the private keys that control the channel are never exposed to untrusted software and hardware, ruling out a large class of potential attacks. These guarantees are robust in the presence of compromises of privileged software, such as the operating system, hypervisor and BIOS. In addition, an attacker who has full control of the hardware off the CPU package, such as RAM, the system bus or the network, cannot violate our security guarantees.

Overall, this paper makes the following contributions: (i) it presents Teechan, a practical framework for low-latency, high-throughput, secure off-chain Bitcoin transactions between mutually-distrusting parties. We describe the background, assumptions, threat model, and the general architecture of this framework in detail; (ii) it describes the detailed operation of a prototype implementation of this framework on the existing Intel SGX CPUs; and (iii) finally, it presents preliminary performance measurements from our prototype implementation, demonstrating that Teechan can achieve 2,480 tx/s on a single payment channel, thereby enabling system-wide aggregate throughput that can compete with and surpass requirements of credit card payment networks.

2 Background

In this section, we provide background on the key technologies that underpin Teechan. We first give a short overview of Bitcoin, explore why it is unable to scale, and then describe Intel SGX's trusted execution environment technology.

2.1 Bitcoin

Bitcoin [28] is a distributed peer-to-peer network that executes a replicated state machine. Each peer, or *node*, in the network maintains and updates a copy of the Bitcoin *blockchain*, an append-only log that contains the transaction history of every account in the network. Users interact with the network by issuing *transactions* to transfer Bitcoins (*BTC*). Valid transactions consume some previously unspent transaction as *inputs* and create a new unspent *output* that can later be used in a new transaction. To spend an *unspent output*, a condition has to be met that is specified by a *locking script*. Typically, a signature matching an address proves that the user spending the output owns the account claiming the funds. More complex locking scripts can be expressed, such as *m-of-n multisig* transactions, where *m* signatures are required out of *n* possible signatures in order to spend the funds; and *timelocked transactions*, which can only be spent after a specified time in the future.

Transactions are appended to the Bitcoin ledger in batches known as *blocks*. Each block includes a unique ID, and the ID of the preceding block, forming a chain. Peers in the network compete to generate and append these blocks to the blockchain. This process, known as *mining*, is computationally expensive and requires solving a cryptographic puzzle. Miners are compensated for their efforts via the *block reward* as well as the *transaction fees* collected from the transactions in that block. The Bitcoin protocol dynamically adjusts the difficulty of the cryptographic puzzle so that a block is appended to the blockchain at an average rate of one block every ten minutes. In cases where there are multiple blocks with the same parent (*forks*), the network adopts the chain with the greatest difficulty.

This novel protocol architecture protects against *double spend* attacks. In such an attack, two conflicting transactions claim the same unspent outputs. The Bitcoin protocol will ensure that the miners will mine at most one of these transactions, and clients of the network will wait for additional succeeding blocks (typically, 6) to guard against forks and reorganizations [3].

Overall, the Bitcoin protocol suffers from two fundamental limitations. First, because it limits the size of each block and the rate of block generation, the network is fundamentally limited in throughput. Second, because the suffix of the blockchain is subject to reorganization, users must wait until their transactions are buried sufficiently deeply, capping the minimum latency.

2.2 Trusted Execution Environments with Intel SGX

Intel's *Software Guard Extensions* (SGX)[18,19,10] enable specially-designated application code to be executed with confidentiality and integrity guarantees. The main abstraction provided by SGX is a *trusted execution environment* known as a secure *enclave* that isolates code and data using trusted hardware mechanisms in the CPU. Assuming the physical CPU package is not breached, SGX *enclaves* are secure against an attacker with physical access to the machine, including access to memory, the system bus, BIOS, OS, and peripherals.

During execution, enclave code and data reside in a region of protected memory called the *enclave page cache* (EPC). When enclave code and data is resident on-chip, in the cache, it is guarded by CPU access controls. Whenever it is flushed to DRAM or disk, it is encrypted. The memory encryption engine encrypts and decrypts cache lines in the EPC as they are written to and fetched from DRAM. Enclave memory is also integrity protected, ensuring that modifications and rollbacks can be detected and the enclave can terminate execution. Only application code executing inside the enclave is permitted to access the EPC. Enclave code can, however, access all memory outside the enclave directly. In addition, as enclave code is always executed in user mode, any interaction with the host OS through system calls, e.g. for network or disk I/O, must execute outside the enclave. Invocations of the code in the enclave can only be performed through well-defined entrypoints under the control of the application programmer.

The critical additional functionality provided by SGX is that of *remote attestation* [20], which enables an enclave to acquire a signed statement from the CPU that it is executing a particular enclave with a given hash of memory, known as a *quote*. An auxiliary service, known as the *Intel Attestation Service* (IAS), can certify that these signed statements originate from authentic CPUs conforming to the SGX specification.

3 Model and Goals

Payment channels are applicable whenever two parties have long-lived financial relationships that involve frequent interaction that needs to be performed with high-throughput, low latency, and privacy guarantees. The central task for Teechan, then, is to construct a duplex payment channel between two such endpoints, equipped with trusted execution environments.

3.1 Threat Model and Assumptions

Our threat model assumes that both parties wish to exchange funds but mutually distrust one another. Each party is potentially malicious, that is, they may attempt to steal funds, avoid making payments and deviate from agreement if it benefits them. Any time during channel establishment, execution and closure, each party may drop, send, record, modify and replay arbitrary messages in the protocol. Either party may terminate the channel at any time, and failures are possible.

We assume that each party runs their own TEE-capable machine and trusts the Bitcoin blockchain, its own environment, the local and remote TEEs, and the code that executes the Teechan duplex channel protocol. The rest of the system, such as the network between the parties and the other party's software stacks (outside the TEE) and hardware are untrusted. During protocol execution, any party may therefore: (i) access or modify any data in its memory or stored on disk; (ii) view or modify its application code; and (iii) control any aspect of its OS and other privileged software. We assume that each party trusts their own environment to not be compromised during channel execution.

Our threat model does not take into account denial-of-service attacks or side-channel attacks. In practice, these are difficult to exploit, possible to mitigate, and the subject of much separate work outside the scope of this paper.

3.2 Goals

A payment channel should operate as follows. A channel is established with a *setup transaction* in the blockchain to which each party deposits an amount as credit. While the channel is open, each party can pay its counterparty via transaction messages sent from the payer to the payee. A payment can only be claimed if it was granted by a party, that is, theft should not be possible. At any point in time, the channel has a balance that ought to reflect the difference between the amounts paid in each direction. The balance should never exceed the credit in either direction. Either party can terminate the channel at any time and settle the balance with a terminating transaction it places in the blockchain. The terminating transaction reflects a balance that comprises all payments made by the terminator and all payments received by the terminator from its counterparty. Failures should only negatively impact the party who failed.

Parties should only need to synchronize with the Bitcoin network during channel establishment and at the point of settlement. In particular, they should not need to monitor the blockchain during the lifetime of the channel.

4 Teechan

The key intuition behind Teechan is to exploit trusted execution environments to act as a trusted third party between two parties, *Alice* and *Bob*.

Broadly, Teechan works as follows. First, at setup, the enclave at each party is securely given mutual secrets belonging to both parties. These secrets can be used at any time to settle the channel, without needing cooperation. Next, while the channel is open, the enclaves maintain channel state internally, free from tampering due to the TEE guarantees. Updates (payments) are performed through a secure interface. Finally, Teechan leverages the enclave's secure execution guarantee to settle the channel at termination. Only on termination does an enclave generate a Bitcoin transaction that can be placed in the blockchain.

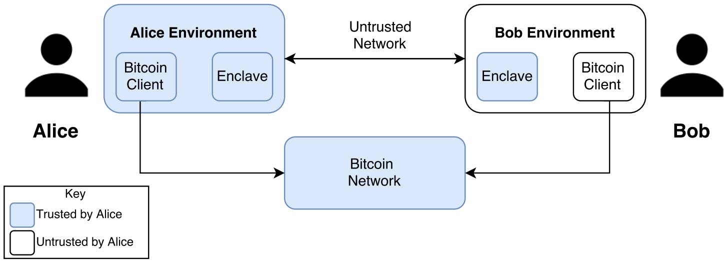

Fig. 1. The Teechan duplex channel architecture. Entities trusted by *Alice* are shaded.

Unlike previous approaches [12,30], Teechan does not make a settlement transaction available until channel termination. The availability of such a transaction is the root cause behind much of the complexity of today's payment channel implementations: it causes race conditions, requires a timely response when leaked to the network prematurely, and requires additional infrastructure for monitoring. Overall, this factoring of crucial channel functionality into trusted enclaves yields a framework that is simple and efficient.

Figure 1 shows the Teechan duplex channel architecture. Both *Alice* and *Bob* run their own TEE *enclaves* alongside a connection to the Bitcoin network. This connection is only used during channel establishment and closure. The figure highlights the entities trusted by *Alice*. An identical figure can be constructed for *Bob* using symmetry of the payment channel.

We next describe the protocol, and informally analyze its security properties in Section 4.2.

4.1 Protocol

The Teechan channel protocol operates in 3 phases: (i) *channel establishment*, (ii) *channel operation*, and (iii) *channel settlement*. Figure 2 shows the messages exchanged during each of these three phases in detail. *Alice*, *Bob*, *Alice's enclave* (denoted $Enclave_A$) and *Bob's enclave* (denoted $Enclave_B$) are modeled as separate entities.

For simplicity, we ignore mining fees in our example, though they are supported in our implementation and affect only the initial setup and the final settlement transactions.

A. Channel establishment In the first phase, Teechan establishes the duplex payment channel between *Alice* and *Bob*. Similar to previous work [17,12,30], we construct a payment channel using *setup* and *refund* transactions. Both *Alice* and *Bob* deposit funds into a *2-of-2 multisig* Bitcoin address, forming a *setup* transaction. A *refund* transaction is constructed that spends the *setup* transaction and returns *Alice* and *Bob*'s deposits back to them. The *refund* transaction is bounded by a lock-time [33], making it valid only starting some time in the future. The channel must be terminated prior to this time, otherwise either party can terminate the channel as if the balance was zero.

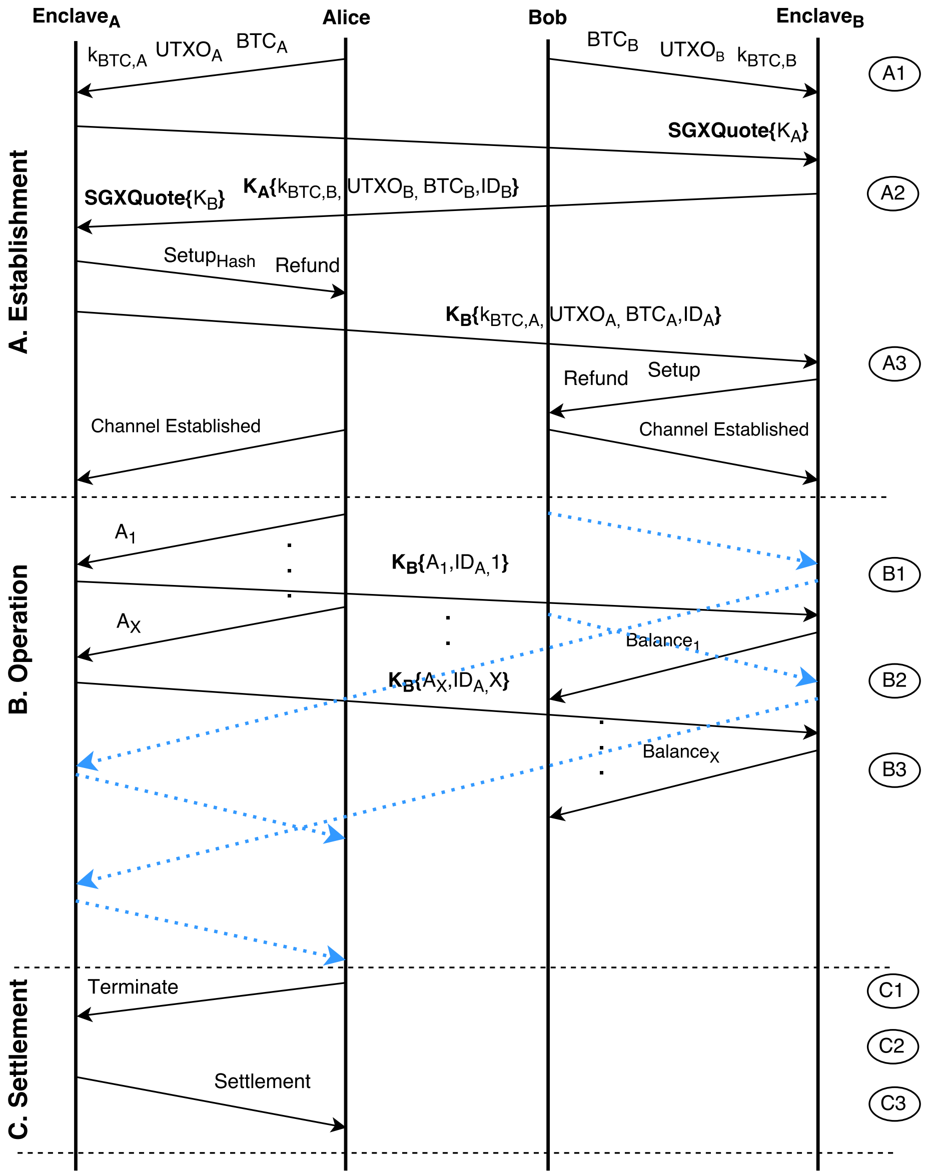

Fig. 2. The Teechan duplex channel protocol. For illustration purposes *Bob*'s payments to *Alice* are shaded, but unlabeled.

A1. First *Alice* and *Bob* each provision their enclaves to construct *setup* and *refund* transactions. This requires: (i) their Bitcoin *private keys* (denoted $k_{BTC,A}$ and $k_{BTC,B}$, respectively), (ii) the *unspent transactions outputs sets* that they wish to include in the *setup* transaction, denoted $UTXO_A$ and $UTXO_B$, respectively, and (iii) the amount to deposit in the *setup* transaction (denoted BTC_A and BTC_B, respectively).

A2. Second, $Enclave_A$ and $Enclave_B$ establish a secure communication channel, authenticating each other through remote attestation. To achieve this, each enclave generates an asymmetric encryption key pair and a random secret key using a secure random number generator. Alice's enclave binds the generated asymmetric public key to a quote, and she sends it to Bob. Bob can then verify that any message he encrypts under K_A can be decrypted solely by Alice's enclave, and that that the enclave is running the desired enclave code with the requisite binary hash. The same is done in the reverse direction, so Alice's Enclave obtains Bob's public key. Upon successful mutual verification, $Enclave_A$ and $Enclave_B$ know that any data encrypted under K_A and K_B can only be read by the opposite enclave, which allows a confidential communication channel to be established.

A3. Bob's enclave then presents its random secret key (denoted ID_B), along with Bob's *setup* data it received in step A1., to Alice's enclave. Alice's enclave generates the signed *setup* and *refund* transactions internally, and reveals to Alice the hash of the *setup* transaction, denoted $Setup_{Hash}$, as well as the *refund* transaction. Only Alice's Enclave knows the *setup* transaction at this point.
Alice then presents its random secret key ID_A, along with Alice's *setup* data it received in step A1., to Bob's enclave. Bob's enclave generates the *setup* and *refund* transactions internally, and reveals both to Bob. Bob then broadcasts the *setup* transaction onto the blockchain, establishing the channel. Alice is notified of channel establishment by noting a transaction matching $Setup_{Hash}$ on the blockchain.
At the end of this three-step handshake, a secure communication channel is established between the two enclaves. The slight asymmetry of the three-step handshake is critical for achieving the termination and loss properties, as described in Section 4.2.

B. Channel operation Once a channel has been *established* between $Enclave_A$ and $Enclave_B$, *Alice* and *Bob* can begin exchanging funds. In this phase, neither *Alice* nor *Bob* need to maintain a connection with the Bitcoin network. They can rapidly make transactions through peer-to-peer updates. Note that in figure 2 payments made from *Bob* to *Alice* are shaded, but unlabeled. This is for illustration purposes only and these payments exhibit the exact same behavior, only in a symmetric fashion to the payments sent from *Alice* to *Bob*.

B1. In order to send funds to Bob, *Alice* sends a request locally to her enclave, specifying the amount of Bitcoin she wishes to transfer to Bob. These requests are denoted A_1 through A_X, representing arbitrarily many payment requests.

B2. When an enclave receives a payment request from the enclave owner, it first checks that the resultant balance is below the sender's deposit. If so, it updates the balance and crafts a message authorizing the payment. The message contains the random secret key of the paying enclave ID_A and the updated monotonic counter value. The message is encrypted under the appropriate asymmetric public key K_B. Alice sends this message to Bob.

B3. Bob receives the message and sends it to his enclave. Once the enclave receives the message it decrypts it and asserts that it contains the correct secret key and that the value of the counter is greater by one than the previously presented counter. Then it updates the balance and the counter for incoming messages. Finally it notifies Bob of the new balance.

Note that each party, outside the enclave, is in charge of maintaining a reliable FIFO channel for the payment messages. This can be achieved with standard go-back-n or similar protocols.

C. Channel settlement The final stage of the Teechan protocol is *channel settlement*. In this phase, the payment channel is closed, and a valid transaction settling the balance between *Alice* and *Bob* is broadcast to the Bitcoin network, thus releasing the funds in the *setup* transaction.

C1. At any point during phase 2, either party may send a *terminate* request to their enclave.

C2. Once an enclave receives a *terminate* request from its owner, it generates a *settlement* transaction signed with $k_{BTC,A}$ and $k_{BTC,B}$ that spends the funds held in the *setup* transaction according to the current channel balance. It returns this transaction to the host, destroys all state held in enclave memory and halts its execution.

C3. The party then forwards this to the Bitcoin network to complete the settlement.

Unlike prior art, Teechan does not suffer from channel exhaustion, and can continually reuse the channel as long as the funds are moving back and forth between the parties. In contrast, prior art constructs duplex channels out of two simplex channels, where the participants can send at most a pre-agreed amount. Exhaustion forces channel resets even when the participants have not exceeding channel capacity, can force termination of the channel, and requires unilateral settlement transactions to consist of multiply nested Bitcoin transactions. In contrast, Teechan enables infinite channel reuse: Alice and Bob can send funds back and forth until channel timeout.

Termination of a channel at the end of its lifetime is identical to prior work. When the *refund* transaction becomes valid, either party can choose to broadcast the *refund* transaction, or to settle the current state of the channel, as described above. Whichever transaction is confirmed by the Bitcoin network dictates the outcome of the channel.

Note 1. Note that a unilateral termination of the channel by Bob cannot harm Alice: he will not be able to receive further payments from Alice, but the closed channel will accurately reflect all payments Alice is aware of. If Bob fails to broadcast the termination transaction to the Bitcoin network, Alice can independently close it from her side.

Teechan is not a consensus protocol, nor is it designed to solve the Byzantine-Generals Problem: Alice and Bob might not agree on the termination state, but Alice's termination state is guaranteed to be acceptable by Bob, and vice-versa.

4.2 Security

In this section, we provide the intuition behind the security properties of the protocol; we defer formal proofs of security to the full paper.

Any time during channel establishment, execution and closure, each party may drop, send, record, modify and replay arbitrary messages in the protocol. As such, we informally evaluate and discuss the security of our protocol against malicious and misbehaving parties. Note that any external adversary in the system, such as an actor who has compromised the network, has fewer privileges than the counterparty in the channel, and can be so subsumed by a malicious counterparty. Therefore, arguing security against the opposite party in a channel is strong enough to protect against any external adversary.

During channel establishment, each enclave is provisioned with sensitive *setup* data from both parties. This is always performed through a secure interface, encrypted with a key internal to the enclave. Communication with the counter party's enclave is only performed after verifying it is indeed an enclave running the Teechan code. Finally, no party can access the *setup* transaction before the other party has the *refund* transaction. Therefore, at the end of channel establishment both parties have the refund transaction and only the enclaves have both secrets.

During channel operation, once a party receives a payment channel, the sending party's enclave has already registered this payment. Therefore, and due to the counter encoded in each payment message, a party cannot revert a payment it has made when settling the channel. Early termination can only prevent a party from receiving future payments, not harming the other party.

Intel SGX We implement Teechan on Intel SGX, which has two particular behaviors that need special attention.

Note 2. First, while SGX protects running enclaves against replay attacks, and allows snapshots of enclave state to be stored in non-volatile memory securely, it cannot protect saved snapshots from rollback attacks. This is because current Intel SGX processors do not provide hardware monotonic counters. These are required to prevent a stale enclave snapshot from being replayed. Replay attacks are detrimental to Teechan security: if Alice could revert the system to an old state, she could take a snapshot when the balance is in her favor, and after sending payments to Bob, revert to that old state and settle the channel at a wrong balance. Teechan is therefore not crash-failure resilient. If Alice fails, she can either ask Bob to settle at the current balance, or wait until the refund transaction becomes available.

Note 3. The validity of an Intel SGX attestation is certified through the Intel Attestation Service (IAS), which ensures that the quote originated from a genuine SGX CPU. In our prototype, we do not use a trusted connection between the enclave and the IAS; the quote is verified in untrusted code, executed by the owner of the enclave during the setup phase. This is benign because misbehavior by a party at this stage would only harm that party, as it would expose their private keys to a fraudulent remote enclave. Terminating the SSL connection to IAS inside the enclave [38] would avoid this situation, but it would just bloat the size of the trusted computing base.

Furthermore, we note that IAS is a not a necessary principle for remote attestation, but an artificial requirement for the current version of Intel SGX processors.

5 Implementation & Evaluation

We have implemented Teechan using Intel SGX as a TEE and deployed it on the Bitcoin testnet for evaluation. All of our implementation is fully compatible with the standard Bitcoin network. We report here on some practical elements of the construction and evaluation results.

Teechan Implementation The Teechan prototype contains two components: a Bitcoin client and an Intel SGX enclave application that executes the secure Teechan protocol. Each party in the payment channel maintains and executes their own client and enclave. For the Bitcoin client, we fork the open-source `libbitcoin-explorer` [23], a C++ Bitcoin library that communicates with the Bitcoin network. `libbitcoin-explorer` relays transactions and requests to a `bitcoin-server` [24], a full Bitcoin peer in the Bitcoin network. In our experiments we use `libbitcoin-explorer 3.0.0` and communicate with a set of public-facing `bitcoin-servers` [34].

For the Teechan enclave application, we ported a subset of `Bitcoin Core 0.13.1` [2] to the Intel SGX. Only several features of the Bitcoin core are needed inside the enclave: (i) multisig address generation; (ii) transaction creation; (iii) transaction signing; and (iv) signature verification. For asymmetric encryption between enclaves, we use RSA with 4096-bit keys, implemented by an SGX-compatible fork of the cryptographic library, `mbedTLS 2.2.1` [37]. Both `libbitcoin-explorer` and the Teechan enclave communicate over TCP, through a lightweight message queuing library, `ZeroMQ 4.2.1` [36].

Experimental Setup To evaluate Teechan, we run all experiments on a single machine, forming a channel between two parties that communicate through sockets.

We use an SGX-enabled 4-core Intel Xeon E3-1280 v5 at 3.70`GHz` with 64`GB` of RAM, Ubuntu 14.04 LTS with Linux kernel 3.19. Each party is bounded to two cores. We deactivate hyper-threading, compile the applications using GCC 5.4.0 with `-O2` optimizations and use the Intel SGX SDK 1.5.

Performance We measure the time it takes Teechan to perform each of the three phases of the protocol. To measure the throughput of our prototype, we simulate an exchange between two parties where each party sends and receives messages sequentially in a loop. At each iteration of the loop, a party can choose uniformly at random whether to wait for a transaction or send a transaction. We measure the time it takes 10 million transactions to be exchanged. Our experiments results in an upper bound, as it eliminates network bandwidth and latency. We defer a thorough evaluation of Teechan under varying network conditions, enclave topologies, and transaction patterns to the full version of the paper.

Channel establishment and final settlement times are bounded by the time to place the transactions in the blockchain. Once the channel is set up, we measure an average latency of 0.40ms and average throughput of 2480tx/s.

For the purpose of demonstration, we provide a reference to a Teechan payment channel that was established, operated, and settled on the Bitcoin test network. Each side deposited 50 bitcoin in the *setup* transaction[3], and the channel was closed with a balance of 9 bitcoin for Bob[4]. Note that fees of 0.002 Bitcoin were paid on both transactions.

6 Related Work

Direct payments were proposed by [7] to allow direct privacy-preserving payments while maintaining privacy. Their assumptions are significantly weaker than those offered by payment channels. For example, cheating is only enforced in retrospect, through punishment mechanisms external to the system.

The performance issues of the blockchain network were addressed by various proposals, from the GHOST protocol and alternatives to the chain structure [31,22,32], to alternative block generation techniques [14,21,29]. Others [26,5,27] build on classical consensus protocols [6] or operate in permissioned settings. While they all improve on the Nakamoto blockchain performance, none can reach the performance offered by direct channels that do not require global system consensus on each transaction.

Bitcoin *micropayment channels* were first discussed by Hearn and Spilman [17]. They could not be deployed directly as they require changes to the Bitcoin protocol, unlike Teechan.

Decker and Wattenhofer [13] realize Duplex Micro-payment Channels under the same restriction (changes to Bitcoin). Instead of forming two independent one-way channels, they propose a construction that allows two parties to form a pair of channels, one in each direction, and re-balance them as needed, that is, when the credit in one direction is depleted but after there have been transactions in the opposite direction. However, the number of resets possible is limited at channel construction, depending on the time allotted for the refund timeout and the bound on the time to place a transaction on the blockchain. Therefore the total amount that can be sent on the channel in one direction is bounded by the deposit amount times the maximal number of resets. Additionally, on disagreement, multiple transactions have to be placed on the blockchain, the exact number depends on the number of resets allowed on the channel. In Teechan there is no limit on the total amount moving in any direction, and only two transactions are ever placed in the blockchain.

The Lightning Network [30] allows for unlimited reuse of its channels by having the two parties form a series of transaction structures, where each update invalidates the previous one. If a party tries to settle the channel on the blockchain with an invalidated state, its counterpart sees this transaction on the blockchain and can redirect all the deposited amount to itself. The performance impact of this protocol is that payments happen in a serial fashion, one at a time. Updating the balance takes about four message exchanges (from deciding on the new value to sending transaction signatures in a certain order). During these exchanges, no payments can be reliably made. In Teechan a payment is done with a single message, and payments in both directions can be made

[3] http://tbtc.blockr.io/tx/info/d55dcfebff45d7e4f9970edd053c87cb0b659e459f4f6360d4a2c17837e79410

[4] http://tbtc.blockr.io/tx/info/1a736822a4f518eb137658030f1e11a804d64d1da48c195222f604aaf2df908e

concurrently, making it full-duplex rather than half-duplex. Additionally, Teechan does not require monitoring the blockchain, as the principals never control a transaction that reflects an old state. The SGX issues only one settlement transaction and stops. An attacker gains no advantage from publishing such a transaction unilaterally.

The Lightning Network effort aims to construct a multi-hop Layer 2 network of payment channels, a topic that is outside the scope of this report. We believe the LN network construction can be made to work with Teechan channels, a challenge we defer to future work.

7 Conclusion

We presented Teechan, full-duplex payment channels based on the existing Bitcoin network with trusted execution environments. The Teechan prototype, built on Intel SGX, achieves 2,480 tx/s and a transaction latency of 0.40ms in optimal conditions. It advances the state of the art by obviating the need to modify the underlying Bitcoin protocol for a practical deployment, improving channel performance, and reducing blockchain overhead.

References

1. ANDRESEN, G. Increase maximum block size (bip 101). `https://github.com/bitcoin/bips/blob/master/bip-0101.mediawiki`, retrieved Dec. 2016.
2. BITCOIN COMMUNITY. Open source bitcoin client software. `https://github.com/bitcoin/bitcoin`.
3. BITCOIN COMMUNITY. Some things you need to know. `bitcoin.org/en/you-need-to-know`.
4. BLOCKCHAIN.INFO. Confirmed transactions per day. https://blockchain.info/charts/n-transactions?timespan=all. Accessed on 2016-12-15.
5. CACHIN, C. Architecture of the Hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers* (2016).
6. CASTRO, M., LISKOV, B., ET AL. Practical byzantine fault tolerance. In *OSDI* (1999), vol. 99, pp. 173–186.
7. CHAUM, D. Blind signatures for untraceable payments. In *Advances in Cryptology* (1983), vol. 82, pp. 199–203.
8. COINDESK.COM. Is segregated witness the answer to bitcoin's block size debate? `http://www.coindesk.com/segregated-witness-bitcoin-block-size-debate/`, retrieved December 2016.
9. COMMUNITY, B. Scalability. https://en.bitcoin.it/wiki/Scalability. Accessed on 2016-12-15.
10. COSTAN, V., AND DEVADAS, S. Intel SGX Explained. Tech. rep., Cryptology ePrint Archive, 2016.
11. CROMAN, K., DECKER, C., EYAL, I., GENCER, A. E., JUELS, A., KOSBA, A., MILLER, A., SAXENA, P., SHI, E., AND SIRER, E. G. On scaling decentralized blockchains. In *Proc. 3rd Workshop on Bitcoin and Blockchain Research* (2016).
12. DECKER, C., AND WATTENHOFER, R. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems* (2015), Springer.
13. DECKER, C., AND WATTENHOFER, R. A fast and scalable payment network with Bitcoin Duplex Micropayment Channels. In *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings* (2015), Springer, pp. 3–18.
14. EYAL, I., GENCER, A. E., SIRER, E. G., AND VAN RENESSE, R. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (2016), pp. 45–59.
15. GARZIK, J. Block size increase to 2mb (bip 102). `https://github.com/bitcoin/bips/blob/master/bip-0102.mediawiki`, retrieved Dec. 2016.
16. GARZIK, J. Making decentralized economic policy. `http://gtf.org/garzik/bitcoin/BIP100-blocksizechangeproposal.pdf`, retrieved December 2016.
17. HEARN, M., AND SPILMAN, J. Bitcoin contracts. https://en.bitcoin.it/wiki/Contracts. Accessed on 2016-12-15.
18. INTEL. Product Change Notification. `https://qdms.intel.com/dm/i.aspx/5A160770-FC47-47A0-BF8A-062540456F0A/PCN114074-00.pdf`, October 2015.
19. INTEL CORP. Software Guard Extensions Programming Reference, Ref. 329298-002US. `https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf`, 2014.

20. JOHNSON, SIMON ET AL. Intel® Software Guard Extensions: EPID Provisioning and Attestation Services. https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services, 2016.
21. KOGIAS, E. K., JOVANOVIC, P., GAILLY, N., KHOFFI, I., GASSER, L., AND FORD, B. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)* (2016), pp. 279–296.
22. LEWENBERG, Y., SOMPOLINSKY, Y., AND ZOHAR, A. Inclusive block chain protocols. In *Financial Cryptography* (Puerto Rico, 2015).
23. LIBBITCOIN COMMUNITY. Bitcoin explorer. https://github.com/libbitcoin/libbitcoin-explorer.
24. LIBBITCOIN COMMUNITY. Obelisk: Bitcoin full node and query server. https://github.com/libbitcoin/libbitcoin-explorer.
25. LOMBROZO, E., LAU, J., AND WUILLE., P. Bip 141 segregated witness. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki. Accessed on 2016-12-15.
26. MAZIERES, D. The Stellar consensus protocol: A federated model for Internet-level consensus. https://web.archive.org/web/20161025142145/https://www.stellar.org/papers/stellar-consensus-protocol.pdf, 2015.
27. MILLER, A., XIA, Y., CROMAN, K., SHI, E., AND SONG, D. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2016), CCS '16, ACM, pp. 31–42.
28. NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf, 2008.
29. PASS, R., AND SHI, E. Hybrid consensus: Efficient consensus in the permissionless model. Cryptology ePrint Archive, Report 2016/917, 2016. http://eprint.iacr.org/2016/917.
30. POON, J., AND DRYJA, T. The bitcoin lightning network: Scalable off-chain instant payments. Tech. rep., Technical Report (draft 0.5.9.1). https://lightning.network. Retreived Dec. 2016., 2016.
31. SOMPOLINSKY, Y., AND ZOHAR, A. Accelerating Bitcoin's transaction processing. fast money grows on trees, not chains. In *Financial Cryptography* (Puerto Rico, 2015).
32. THE ETHEREUM COMMUNITY. Ethereum white paper. https://github.com/ethereum/wiki/wiki/White-Paper, retrieved July. 2015.
33. TODD, P. Allow transactions to be made unspendable until sometime in the future (bip 65). https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki, retrieved December 2016.
34. UNSYSTEM COMMUNITY. Public obelisk servers. https://wiki.unsystem.net/en/index.php/Obelisk/Servers.
35. WUILLE, P. Block size following technological growth (bip-sipa). https://gist.github.com/sipa/c65665fc360ca7a176a6, retrieved December 2016.
36. ZEROMQ.ORG. Distributed messaging library. https://github.com/zeromq/libzmq.
37. ZHANG, F. mbedtls-sgx: a sgx-friendly tls stack (ported from mbedtls). https://github.com/bl4ck5un/mbedtls-SGX.
38. ZHANG, F., CECCHETTI, E., CROMAN, K., JUELS, A., AND SHI, E. Town crier: An authenticated data feed for smart contracts.