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 and the Initiative for CryptoCurrencies and Contracts (IC3)

Abstract. Blockchain protocols are inherently limited in transaction throughput and latency. Recent efforts to address performance 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 implementation requires changes to the underlying protocol and the 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. It: (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 at most 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, not counting network latencies, Teechan can achieve 2,480 transactions per second on a single channel, with sub-millisecond latencies.

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 industries and applications. As a result, adoption has grown rapidly, leading to an exponential increase in the number of transactions sent per day [3]. This growth exacerbates a natural problem: the Nakamoto consensus protocol that underpins Bitcoin is fundamentally limited in transaction throughput and imposes a significant minimum transaction latency [9]. 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.

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 maximum of 7 tx/s [33]. Recent proposals have suggested either tuning these parameters, such as increasing the block size or reducing the block interval [13, 1, 12, 37]; or modifying the protocol, for example, by incrementally creating blocks so as to avoid centralization bottlenecks and increase throughput [11]. The

former approach cannot scale Bitcoin by more than one order of magnitude, while the latter requires changes to the underlying protocol, which practitioners have been reticent to make. Other research suggests that hardware limits, such as the cost of signature verification and storage latencies, cap Bitcoin to 200 tx/s [10].

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* [14, 10, 29]. Payment channels allow for efficient, trustless fund transfers, in which parties can exchange transactions without having to impact 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 bidirectional payment channels, none have been deployed securely on the existing Bitcoin network, as they assume modifications to the underlying protocol. Specifically, they require transaction IDs to be set before they are signed—a proposal to address this issue, SegWit [24], is currently mired in controversy [7].

We present **Teechan**, the first *high-performance micropayment protocol* that supports practical, secure, and efficient fund transfers on the current Bitcoin network. Similar in design to Duplex Micropayment 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) to strengthen the guarantees provided by the framework: (i) Teechan does not require any changes to the Bitcoin network; (ii) it enables infinite channel reuse as long as the balance does not exceed the channel limits; and (iii) it is time- and space-efficient, requiring only one-way messages for sending payments and two transactions to be placed on the blockchain in total. Section 6 discusses the comparison to prior art in detail.

At a high level, current implementations of TEEs can provide confidentiality and integrity guarantees for code and data, but cannot guarantee liveness or safe termination for a protocol. Teechan is designed in a manner that, despite these limitations, no party can gain access to more funds than their current net balance. In particular, the TEE ensures that the private keys that control the channel are never exposed to untrusted software or hardware, ruling out a large class of potential attacks. These guarantees are robust in the presence of compromised privileged software, such as the operating system, hypervisor, and BIOS. In addition, an attacker who has full control of the hardware outside of the CPU package, including the RAM, the system bus and the network, cannot violate our security guarantees.

Overall, our 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; (ii) it describes the detailed operation of a prototype implementation of this framework using Intel SGX as the TEE; and finally, (iii) 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 the requirements of credit card payment networks.

2 Background

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

Bitcoin Bitcoin [27] 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 unspent transactions as *inputs* and create new unspent *outputs* that can later be used in a new transaction. To spend an *unspent output*, a condition specified by a *locking script* must be met. 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 to spend the funds; and *timelocked transactions*, which can only be spent after a point 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 in which there are multiple blocks with the same parent (*forks*), the network adopts the chain with the greatest difficulty.

This 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 [34].

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, incurring a minimum latency.

Trusted Execution Environments with Intel SGX Intel's *Software Guard Extensions* (SGX)[15, 18, 8] enable application code to be executed with confidentiality and integrity guarantees. SGX provides *trusted execution environments* known as secure *enclaves* that isolate code and data using hardware mechanisms in the CPU. Assuming

the physical CPU package is not breached, SGX enclaves are protected from an attacker with physical access to the machine, including access to the memory, the system bus, BIOS, 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, it is guarded by CPU access controls; when it is flushed to DRAM or disk, it is encrypted. A 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 code executing inside the enclave is permitted to access the EPC. Enclave code can, however, access all memory outside the enclave directly. 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 enclave code can only be performed through well-defined entrypoints under the control of the application programmer.

In addition, SGX supports *remote attestation* [19], 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*. A third-party attestation service, e.g., as provided by 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 when two parties have long-lived financial relationships that require frequent interaction with high-throughput, low latency, and privacy guarantees. The central goal for Teechan, then, is to construct a duplex payment channel between two such endpoints, assuming that these endpoints are equipped with trusted execution environments.

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, i.e., they may attempt to steal funds, avoid making payments, and deviate from the 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 has a 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.

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 separate work outside the scope of this paper.

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 must 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 that 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 intuition behind Teechan is to exploit trusted execution environments (TEEs) to act as a trusted third party between two parties, *Alice* and *Bob*.

At a high-level, Teechan works as follows. First, at setup, the TEE 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 TEEs maintain channel state internally, free from tampering due to the guarantees of trusted execution. Updates (payments) are performed through a secure interface. Finally, Teechan leverages secure execution to settle the channel at termination. Only on termination does a TEE generate a Bitcoin transaction that can be placed in the blockchain.

Unlike previous approaches [10, 29], 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. This factoring of crucial channel functionality into TEEs yields a simple and efficient approach.

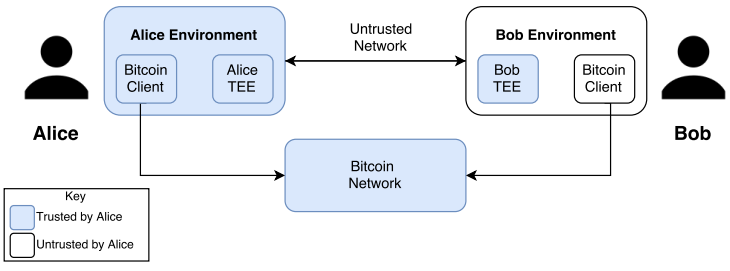

Fig. 1. Teechan Duplex Channel Architecture (Entities trusted by *Alice* are shaded.)

Figure 1 shows the Teechan duplex channel architecture. Both *Alice* and *Bob* run their own TEEs 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 channel. We next describe the protocol, and informally analyze its security in Section 4.2.

4.1 Protocol

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

For simplicity, we ignore mining fees in our example, although 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 prior work [14, 10, 29], 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 using the nLockTime [31] transaction field, 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 no transactions took place.

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

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

A3. TEE_B then presents its random secret key (denoted ID_B), along with Bob's *setup* data that it received in step A1, to TEE_A. A signature over this message, under the private key of TEE_B (denoted $Sigk_B$), is also presented to ensure that it came from TEE_B. TEE_A 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 TEE_A knows the *setup* transaction at this point.

TEE_A then presents its random secret key ID_A, along with Alice's *setup* data that it received in step A1 and the corresponding signature $Sigk_A$, to TEE_B. TEE_B 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.
Note that Bob could maul the *setup* transaction before broadcasting it, making Alice's *refund* transaction invalid. In this case, Alice presents the mauled *setup* transaction to TEE_A to issue a new *refund* transaction that closes the channel immediately. This requires that keys should never be re-used between separate channels, as is already a recommended good practice. In Teechan, mauling the *setup* transaction is equivalent to a denial-of-service attack.

At the end of this three-step handshake, a secure communication channel is established between the two TEEs. The slight asymmetry of the handshake is critical for achieving the termination and loss properties described in Section 4.2.

B. Channel operation Once a channel has been *established* between TEE_A and TEE_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, for illustration purposes only. These payments exhibit the same behavior, in a symmetric fashion, to the payments sent from *Alice* to *Bob*.

B1. To send funds to Bob, *Alice* sends a request locally to TEE_A, specifying the amount of Bitcoin that she wishes to transfer to Bob. These requests are denoted A_1 through A_X, representing arbitrarily many payment requests.
B2. When a TEE receives a payment request from the owner, it first checks that the current balance is greater than the amount to send. If so, it updates the balance and generates a message authorizing the payment. The message contains the random secret key of the paying TEE 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 TEE_B. Once the TEE 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 TEE, 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. Tampering with the order of messages is equivalent to a denial-of-service attack on the recipient only; the sender always processes a payment. It is therefore in the best interest of the receiver to ensure a reliable FIFO channel.

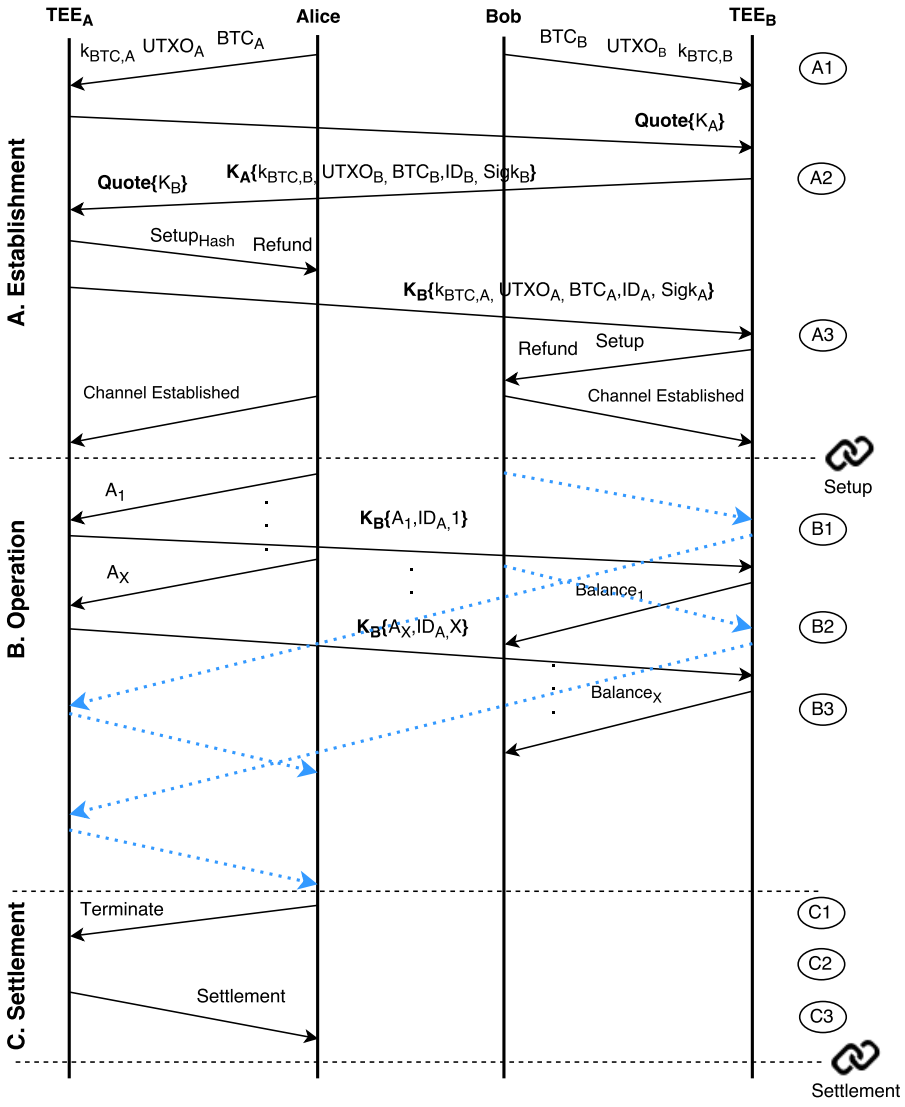

Fig. 2. Teechan Duplex Channel Protocol (For illustration purposes *Bob*'s payments to *Alice* are shaded, but unlabeled.)

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 B, either party may send a *terminate* request to their TEE.

C2. Once a TEE receives a *terminate* request from its owner, it generates a *settlement* transaction signed with $k_{BTC,A}$ and $k_{BTC,B}$, which 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 TEE memory and halts its execution.
C3. The party then forwards this to the Bitcoin network to complete the settlement.

Similar to the Lightning Network [29], Teechan payment channels do not suffer from channel exhaustion. 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 also similar 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 that a unilateral channel termination 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 of which Alice is aware. 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 may 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 attacker who has compromised the network, has fewer privileges than the counterparty in the channel, and so can be subsumed by a malicious counterparty. Arguing security against the opposite party in a channel is strong enough to protect against any external adversary.

During channel establishment, each TEE is provisioned with sensitive *setup* data from both parties. This is always performed through a secure interface, encrypted with a key internal to the TEE. Communication with the counter party's TEE is only performed after verifying that it is indeed a TEE executing 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 TEEs have both secrets.

During channel operation, once a party receives a payment, the sending party's TEE has already registered this payment. Therefore, and due to the counter encoded in each payment message, a party cannot revert a payment that 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. Intel SGX provides secure TEEs offering both execution integrity and confidentiality against an attacker on the same machine,

even one with physical access. These hardware guarantees, coupled with Teechan's architecture, enable the resulting system to be resilient against an array of attacks. The tight integration of SGX with the CPU ensures that the cost to launch an attack, or even gather enough know-how to attempt one, are orders of magnitude higher than the value expected to be stored in payment channels. Given the current market share of Intel CPUs, users already implicitly trust a single hardware manufacturer with their secret keys. We repeat, however, that nothing in the Teechan protocol is Intel specific, and our protocol can be ported easily to, for example, a Ledger hardware security module [2].

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. SGX protects running enclaves against replay attacks by protecting persistently stored snapshots from rollback attacks through non-volatile hardware monotonic counters, which prevent a stale enclave snapshot from being reused.

In our Teechan prototype, if Alice fails, she can either ask Bob to settle at the current balance, or wait until the refund transaction becomes available. It is straightforward to extend the prototype such that the enclaves persist their state to secondary storage, encrypted under a key and stored with a non-replayable version number from the hardware monotonic counter. Our current implementation, at the time of writing, does not leverage hardware monotonic counters, because, while the counters are fully supported by the existing hardware under Windows [17], the current SGX Linux SDK [16] does not expose them yet. Porting our protocol to Windows or support for monotonic counters in the Linux SDK can address this.

Currently, 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-capable Intel 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 TLS connection to the IAS inside the enclave [39] would avoid this issue, but it is unnecessary under our trust model and would needlessly increase the trusted computing base.

5 Implementation & Evaluation

We evaluate Teechan using Intel SGX on the Bitcoin testnet. Our implementation is fully compatible with the standard Bitcoin network. We report preliminary measurements from this implementation to illustrate the range of achievable performance.

Teechan Implementation The Teechan prototype has 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` [22], a C++ Bitcoin library that communicates with the Bitcoin network. `libbitcoin-explorer` relays transactions and requests to a `bitcoin-server` [23], a full Bitcoin peer in the Bitcoin network. In our experiments, we use `libbitcoin-explorer` version 3.0.0 and communicate with a set of public-facing `bitcoin-servers` [36].

For the Teechan enclave application, we port a subset of `Bitcoin Core` version 0.13.1 [32] to Intel SGX. Only some features of 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. Both `libbitcoin-explorer` and the Teechan enclave communicate over TCP using a lightweight message queuing library (`ZeroMQ` version 4.2.1 [38]).

Experimental Setup To evaluate Teechan, we run all experiments on a single machine, which forms a channel between two parties communicating through network sockets. We use an SGX-enabled 4-core Intel Xeon E3-1280 v5 at 3.7 GHz with 64 GB of RAM, and Ubuntu 14.04 with Linux kernel 3.19. We deactivate hyper-threading, compile the applications using GCC 5.4.0 with `-O2` optimizations and use the Intel SGX SDK 1.7.

Performance We measure the time taken by Teechan to perform each of the three phases of the protocol. To measure the throughput, we emulate an exchange between two parties in which each party sends and receives payments sequentially in lock-step. We measure the time for 10 million transactions to be exchanged. These measurements yield an upper bound for our current implementation, as they eliminate 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 0.40 ms and an average throughput of 2480 tx/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]. A fee of 0.002 bitcoin was paid on each transaction.

6 Related Work

Direct payments were proposed by Chaum [6] in ecash to achieve privacy. The ecash assumptions are significantly weaker than those offered by payment channels. Mainly, cheating is enforced in retrospect, through external punishment mechanisms.

Several proposals address the performance issues of the Bitcoin network, from the GHOST protocol and alternatives to the chain structure [30, 21, 35], to alternative block generation techniques [11, 20, 28]. Others [25, 4, 26] build on classical consensus protocols [5] 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 for each transaction.

Unidirectional Bitcoin *micropayment channels* were first informally discussed by Hearn and Spilman [14]. These could not be deployed directly as they require changes to the Bitcoin protocol, unlike Teechan. Alternative proposals for unidirectional micropayment channels have been made to avoid these changes, however, all unidirectional payment channels only operate in a single direction and suffer from channel exhaustion.

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

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

Decker and Wattenhofer [10] were the first to realize duplex micropayment channels (DMC), improving the exhaustion limit. In DMC, two parties 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. DMC also requires changes to Bitcoin.

Additionally, on disagreement, $1 + d + 2$ transactions have to be placed on the blockchain, where d represents the invalidation tree's active branch. In Teechan, there is no limit on the total amount moving in any direction, and only two transactions are ever placed in the blockchain.

Lightning Network (LN) [29] allows for unlimited reuse of its channels. Two parties form a series of transaction structures, in which 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 concurrently, making it full-duplex rather than half-duplex. On disagreement, the Lightning Network places four transactions in the blockchain, while Teechan places only two.

Informal proposals have been sketched to potentially deploy Lightning Network on the Bitcoin network without any changes to the Bitcoin protocol. However, these come with various limitations: a channel can only be funded by a single party, and parties need to monitor the blockchain in order to react to invalidated states. This is not the case for Teechan, however, as both parties can deposit into a Teechan channel, and neither party ever controls a transaction that reflects an old state.

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

Towncrier [39] also uses a TEE but to provide authenticated data feeds for smart contracts.

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, can achieve 2,480 tx/s and a transaction latency of 0.40 ms. 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.

Acknowledgements

This project received funding from the European Unions Horizon 2020 research and innovation programme under the SecureCloud (Grant agreement No. 690111) project.

References

1. ANDRESEN, G. Increase Maximum Block Size (BIP 101). https://github.com/bitcoin/bips/blob/master/bip-0101.mediawiki. Accessed December 2016.
2. BACCA, N. Attestation Redux : Proving Code Execution on the Ledger Platform. https://gist.github.com/sipa/c65665fc360ca7a176a6. Accessed December 2016.
3. BLOCKCHAIN.INFO. Confirmed Transactions Per Day. https://blockchain.info/charts/n-transactions?timespan=all. Accessed December 2016.
4. CACHIN, C. Architecture of the Hyperledger Blockchain Fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers* (2016).
5. CASTRO, M., LISKOV, B., ET AL. Practical Byzantine Fault Tolerance. In *OSDI* (1999), vol. 99, pp. 173–186.
6. CHAUM, D. Blind Signatures for Untraceable Payments. In *Advances in cryptology* (1983), Springer, pp. 199–203.
7. COINDESK.COM. Is Segregated Witness the Answer to Bitcoin's Block Size Debate? http://www.coindesk.com/segregated-witness-bitcoin-block-size-debate/. Accessed December 2016.
8. COSTAN, V., AND DEVADAS, S. Intel SGX Explained. *IACR Cryptology ePrint Archive* (2016).
9. 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 (BITCOIN 2016)*.
10. 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)*.
11. 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 2016)*.
12. GARZIK, J. Block size increase to 2MB (BIP 102). https://github.com/bitcoin/bips/blob/master/bip-0102.mediawiki. Accessed December 2016.
13. GARZIK, J. Making Decentralized Economic Policy. http://gtf.org/garzik/bitcoin/BIP100-blocksizechangeproposal.pdf. Accessed December 2016.
14. HEARN, M., AND SPILMAN, J. Bitcoin Contracts. https://en.bitcoin.it/wiki/Contracts. Accessed December 2016.
15. INTEL. Product Change Notification. https://qdms.intel.com/dm/i.aspx/5A160770-FC47-47A0-BF8A-062540456F0A/PCN114074-00.pdf, 2015.
16. INTEL. Intel Software Guard Extensions (Intel SGX) SDK. https://software.intel.com/sgx-sdk, 2016.
17. INTEL. Intel Software Guard Extensions (Intel SGX) Windows SDK. https://software.intel.com/sites/default/files/managed/b4/cf/Intel-SGX-SDK-Developer-Reference-for-Windows-OS.pdf, 2016.
18. INTEL CORP. Software Guard Extensions Programming Reference, Ref. 329298-002US. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf, 2014.
19. 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.

20. 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)*.
21. LEWENBERG, Y., SOMPOLINSKY, Y., AND ZOHAR, A. Inclusive Block Chain Protocols. In *Financial Cryptography* (Puerto Rico, 2015).
22. LIBBITCOIN COMMUNITY. Bitcoin Explorer. `https://github.com/libbitcoin/libbitcoin-explorer`. Accessed December 2016.
23. LIBBITCOIN COMMUNITY. Obelisk: Bitcoin Full Node and Query Server. `https://github.com/libbitcoin/libbitcoin-explorer`. Accessed December 2016.
24. LOMBROZO, E., LAU, J., AND WUILLE, P. BIP141: Segregated Witness (Consensus layer). `https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki`, 2015.
25. MAZIERES, D. The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus. `https://www.stellar.org/papers/stellar-consensus-protocol.pdf`. 2015.
26. 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 (CCS 2016)*.
27. NAKAMOTO, S. Bitcoin: A Peer-to-Peer Electronic Cash System. `http://www.bitcoin.org/bitcoin.pdf`, 2008.
28. PASS, R., AND SHI, E. Hybrid Consensus: Efficient Consensus in the Permissionless Model. Cryptology ePrint Archive, Report 2016/917.
29. POON, J., AND DRYJA, T. The Bitcoin Lightning Network: Scalable off-chain instant payments. Technical Report (draft 0.5.9.1). `https://lightning.network`. Accessed December 2016.
30. SOMPOLINSKY, Y., AND ZOHAR, A. Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. In *Financial Cryptography* (Puerto Rico, 2015).
31. THE BITCOIN COMMUNITY. nLockTime. `https://en.bitcoin.it/wiki/NLockTime`. Accessed December 2016.
32. THE BITCOIN COMMUNITY. Open Source Bitcoin Client Software. `https://github.com/bitcoin/bitcoin`.
33. THE BITCOIN COMMUNITY. Scalability. `https://en.bitcoin.it/wiki/Scalability`. Accessed December 2016.
34. THE BITCOIN COMMUNITY. Some Things You Need to Know. `https://bitcoin.org/en/you-need-to-know`. Accessed December 2016.
35. THE ETHEREUM COMMUNITY. Ethereum White Paper. `https://github.com/ethereum/wiki/wiki/White-Paper`. Accessed December 2016.
36. THE UNSYSTEM COMMUNITY. Public Obelisk Servers. `https://wiki.unsystem.net/en/index.php/Obelisk/Servers`. Accessed December 2016.
37. WUILLE, P. Block Size following Technological Growth (BIP-sipa). `https://gist.github.com/sipa/c65665fc360ca7a176a6`. Accessed December 2016.
38. ZEROMQ.ORG. Distributed Messaging library. `https://github.com/zeromq/libzmq`. Accessed December 2016.
39. ZHANG, F., CECCHETTI, E., CROMAN, K., JUELS, A., AND SHI, E. Town crier: An Authenticated Data Feed for Smart Contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS 2016)*.