SmarTor: Smarter Tor with Smart Contracts: Improving resilience of topology distribution in the Tor network

Conference Paper · December 2018

DOI: 10.1145/3274694.3274722

Andre Greubel
University of Wuerzburg

Alexandra Dmitrienko
University of Wuerzburg

Samuel Kounev
University of Wuerzburg

SmarTor: Smarter Tor with Smart Contracts

Improving resilience of topology distribution in the Tor network

Greubel Andre
University of Wuerzburg
Wuerzburg, Bavaria, Germany
andre.greubel@uni-wuerzburg.de

Dmitrienko Alexandra
University of Wuerzburg
Wuerzburg, Bavaria, Germany
alexandra.dmitrienko@uni-wuerzburg.de

Kounev Samuel
University of Wuerzburg
Wuerzburg, Bavaria, Germany
samuel.kounev@uni-wuerzburg.de

ABSTRACT

In the Tor anonymity network, the distribution of topology information relies on the correct behavior of five out of the nine trusted directory authority servers. This centralization is concerning since a powerful adversary might compromise these servers and conceal information about honest nodes, leading to the full de-anonymization of all Tor users. Our work aims at distributing the work of these trusted authorities, such increasing resilience against attacks on core infrastructure components of the Tor network. In particular, we leverage several emerging technologies, such as blockchains, smart contracts, and trusted execution environments to design and prototype a system called SmarTor. This system replaces the directory authorities with a smart contract and a distributed network of untrusted entities responsible for bandwidth measurements. We prototyped SmarTor using Ethereum smart contracts and Intel SGX secure hardware. In our evaluation, we show that SmarTor produces significantly more reliable and precise measurements compared to the current measurement system. Overall, our solution improves the decentralization of the Tor network, reduces trust assumptions and increases resilience against powerful adversaries like law enforcement and intelligence services.

CCS CONCEPTS

• **Security and privacy → Pseudonymity, anonymity and untraceability**; *Privacy-preserving protocols*; *Denial-of-service attacks*; • **Networks** → Network privacy and anonymity;

KEYWORDS

Tor Network, Blockchain, Ethereum, Smart Contract, Secure Hardware, Trused Execution Environment, Intel SGX, Security, Privacy

ACM Reference Format:
Greubel Andre, Dmitrienko Alexandra, and Kounev Samuel. 2018. SmarTor: Smarter Tor with Smart Contracts: Improving resilience of topology distribution in the Tor network. In *2018 Annual Computer Security Applications Conference (ACSAC '18), December 3–7, 2018, San Juan, PR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3274694.3274722

1 INTRODUCTION

In a time of continually increasing state surveillance, privacy technologies are more important than ever before. The Tor project offers one privacy solution for anonymous Internet access. In Tor, the so-called *Directory Authorities* (DAs) distribute topology information necessary to build anonymity preserving paths (*circuits*). These DAs collect topology information of the Tor network (*vote*) and follow a consensus protocol to agree on an aggregated version (*consensus*) of this data, which is then published by every DA. In this process, numerical data is aggregated using the (lower) median value. Because of this rule of the majority and as there are only nine DAs, compromising five of them is sufficient to influence topology information arbitrarily. An attacker can abuse this weakness to conceal information about honest network nodes (*relays*) forwarding information in circuits. This attack leads to entirely compromised circuits that offer no privacy guarantees.

With SmarTor, we aim to improve the resilience of the Tor network against attacks on DAs. While attacks targeting DAs might require great technical and financial investment, the potential gain of a successful attack – complete de-anonymization of all of its users – is enormous. The former intelligence community officer and whistleblower Edward Snowden has leaked information that Tor is indeed a high-priority target for the NSA [35]. Furthermore, the FBI has admitted to being behind several attacks on TOR servers [26, 31].

To prevent such attacks, our idea is to distribute trust among more entities. We want to achieve this using a *Smart Contract* (SC). SCs are computer programs stored on blockchains which enable the conditional processing of data in the underlying blockchain. Their code is distributively executed in a secure and verifiable manner without a centralized instance (cf. Section 2). To tamper with information stored in the blockchain, one would have to get more than 50% of the computational power underlying it, which, e.g., for Ethereum [45], would require the power output of a nuclear power plant. Furthermore, due to the peer-to-peer data distribution of blockchains, it is much harder to launch a Denial-of-Service (DoS) attack on such a system. As such, it is possible to store access information of relay nodes (RNs) with the use of a SC which ensures that only the owner of the data may change or delete it.

However, replacing DAs with SCs is not straightforward since the topology information also includes bandwidth measurements of the RNs. Five out of the nine DAs also take the role of a *Bandwidth Authority* (BA) which measure the bandwidth and vote on the measured value. This measurement process is necessary as the probability of a RN being chosen in a circuit is proportional to its bandwidth. With no actual measurements, an attacker could lie

ACSAC '18, December 3–7, 2018, San Juan, PR, USA
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6569-7/18/12…$15.00
https://doi.org/10.1145/3274694.3274722

about the bandwidth of his RNs [1] , in return being assigned more traffic which facilitate de-anonymization attacks [19]. However, SCs cannot perform bandwidth measurements as they lack synchronous communication and their notion of time is limited to seconds or even minutes.

To tackle this problem, we replace the BAs with a distributed set of untrusted *Bandwidth Measurers* (BMs). These BMs execute a measurement script and send the results to the SC which will aggregate them. The measurement script is protected by a *Trusted Execution Environment* (TEE) and redundancy and a reputation score are introduced to spot and exclude any misbehaving BMs. Since the current measurement script called *speedracer* does not produce precise and reliable results (cf. Appendix B), we design and evaluate a different measurement approach for SmarTor.

Contributions Our contributions can be summarized as follows:

- We propose to utilize smart contracts in order to eliminate DAs. This approach increases resilience against strong adversaries, like law enforcement and intelligence agencies aiming to compromise the entire network, since now they would have to take control over 50% of the computing power of the underlying blockchain, instead of a few servers.
- We designed an improved mechanism to measure the bandwidth of Tor relays, which does not rely on trusted parties like BAs, but instead leverages a distributed network of untrusted entities. While untrusted, these entities leverage security extensions of modern processors to obtain trustworthy measurements performed on untrusted platforms. To account for possible compromise of secure hardware, we introduce a reputation system which can detect malicious reports and enables exclusion of compromised entities from further measurements.
- We implemented a measurement method which, especially for slow relays, is far more reliable and precise than the current process. Different from related work, it does not rely on the publication of additional metadata. We evaluated this script with eleven RNs and show that, despite TEE limitations, our results were seven times closer to the advertised value and scattering only half as much as in the current measurement method.

Outline The rest of the paper is structured as follows: In Section 2, we briefly give an overview of the used technologies and related work. In Section 3, we specify our system and adversary model. In Section 4 we explain the design of SmarTor in detail. Section 5 contains a security analysis. In Section 6 we present a proof-of-concept implementation of this system which we evaluate in Section 7. In Section 8 we conclude and provide and outline of our future work.

2 BACKGROUND

In this section, we provide brief descriptions of bandwidth measurements in Tor, Trusted Execution Environments, blockchains, and Smart Contracts.

2.1 Tor Bandwidth Measurements

At the beginning of Tor, DAs would merely trust the bandwidth values reported by the relays as long it is below a specific, publicly known value. However, this approach was vulnerable to attacks based on false reports [3]. The *TorFlow* project [30] introduced Bandwidth Authorities (BAs) which measure and then vote on the actual bandwidth of the relays. The measurement script called *speedracer* divides the network into slices of relays with similar bandwidth and then repeatedly fetches a large file over a two-hop circuit created of relays in this slice. The ratio of average stream capacity of a particular relay to the rest of the slice is then used to adjust the self-reported bandwidth. While this approach is still active in Tor, it has shortcomings. The measurement over two hop circuits produces an influence on the measurement result from similarly sized relays put in the same slice. Furthermore, due to the relay life cycle [14], the time a RN registered with the DAs also influences the measurement result. We present a short analysis of measurement results published in the current consensus in Appendix B.

2.2 Trusted Execution Environments

Trusted Execution Environments (TEEs) protect execution of security-critical application code against an attacker with full control over a system. They enable program execution in isolation from the rest of the system and are equipped with a hardware protected secret enabling *Attestation* and *Sealing*.

Sealing describes the process of storing data to persistent storage in a way that allows only the protected program to reaccess this data. This is done by deriving a *sealing key* from the hardware protected secret which is unique for each protected program. Attestation is used to verify that the program indeed is running in a genuine environment. For this, the TEE creates *attestation data* which describes its state (like loaded program code) and may also include custom data like keys created by the protected program. This data is then signed using an *attestation key* derived from the hardware secret, enabling verification of this data with the associated public key. This signature can be *linkable* enabling entities to detect if two signatures were produced by the same TEE without revealing further identity information.

While those environments can be instantiated using various hardware platforms like TrustZone [1] for ARM processors or Sanctum [11] for RISC-V platforms, we (without loss of generality) instantiated our system using Intel SGX which we will describe in more detail in Appendix A.

2.3 Blockchain and Smart Contracts

A *blockchain* [28] is a decentralized data structure which stores information in an unchangeable and network-wide agreed sequence. Information is divided into blocks consisting of data records and headers chained together as each header includes the hash of the previous block header, preventing changes in blocks. Data can be accessed by users who own *wallets* in the form of *addresses* represented by the hash of a public key. While data can be accessed for free, inserting data using transactions (signed by the *wallet key*) usually results in a *transaction fee.*

Miners verify transactions and include them in new blocks. They follow a peer-to-peer based consensus protocol to reach agreement

[1] Note that, since there have never been more than five BAs, compromising three of them is sufficient to change the amount of traffic an attacker can attract arbitrarily. As of 2018-06-15, there are only four BAs, making it possible to set the bandwidth and measured bandwidth to zero if controlling at least two of them.

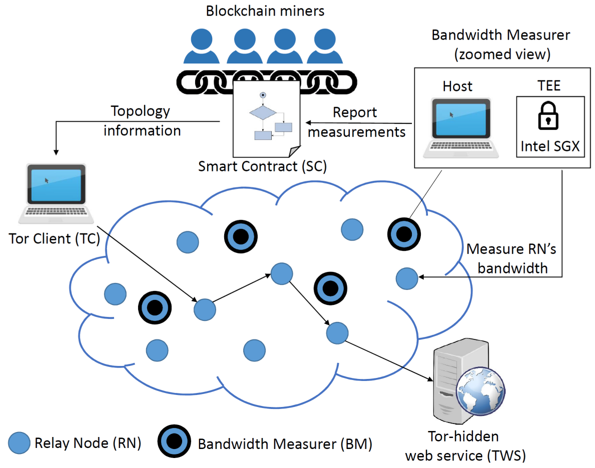

Figure 1: SmarTor System Model

on a single history of data records. Most consensus protocols rely on Proof-of-Work (PoW) algorithms which force them to solve a hard but easily verifiable cryptographic puzzle like partial hash inversion. To achieve consensus, the longest chain is considered as valid and every miner works on the longest chain available. This way, different versions of the blockchain (*forks*) are eventually resolved over time.

Smart Contracts (SCs) are programs stored on a blockchain. They can be called to perform an action by sending a transaction to their address. Executing code is computationally expensive as it has to be done by everyone verifying the corresponding block. Hence, many systems require a fee for the execution of SC code. For instance, Ethereum [45] defines a notion of *gas* which is to be paid with cryptocurrency and needs to be spent in order for the contract to be executed. Limitations of SCs include the lack of more complex data structures or floating point operations even in high-level SC languages, such as Solidity. They furthermore cannot engage in synchronous communication and have no real notion of time apart from the timestamp in the block header whose resolution is limited to seconds or even minutes.

3 SYSTEM AND ADVERSARY MODEL

In this section we first introduce our system model, and then specify our trust assumptions and attacker capabilities.

System Model Our system model is depicted in Figure 1. It includes the following components of Tor's infrastructure: Relay Nodes (RNs), Tor-hidden web services (TWS), and Tor Clients (TCs). In SmarTor, the functionality of Directory (DAs) and Bandwidth Authorities (BAs) is distributed between two types of entities: The Smart Contract (SC) and Bandwidth Measurers (BMs). The SC is a computing program stored on a blockchain, while BMs are managed by volunteers, e.g., by individuals that already operate Tor's relays, by blockchain validators, or by any other users. We assume that BM's platforms are equipped with a hardware-supported Trusted Execution Environment (TEE) where programs can execute security-sensitive operations (e.g., encryption, signing) in isolation from the rest of the system. We will write BM-TEE for the trusted execution environment of a BM.

Inherited Assumptions Trust assumptions about RNs, TWSes, and TCs are inherited from the current Tor system – TCs and TWSes are considered as trusted, while some (but not the majority) of the RNs can be malicious. Like the current Tor browser, the code executed by TCs, SCs or TEEs is developed by a trusted entity and can be viewed and accessed by anybody, once it is published. As usual, there is no trust in the SC, but we assume that no attacker can perform a 50%-attack on the underlying blockchain.

BM Assumptions We moreover make the following assumptions about BMs. The host platform of the BM (BM-Host) is untrusted. However, we trust the vendor of the TEE to ship appropriate secure hardware and support it with Public Key Infrastructure (PKI). Once commissioned, some of the BM-TEEs can be compromised, i.e., their hardware secret can get leaked. However, we assume that the number of compromised BM-TEEs is limited.

Note that hardware attacks against secure hardware are technically involved and costly. Associated attacks cost up to 1, 000, 000$ when dealing with tamper-resistant devices like smartcards [38]. While successful hardware attacks against Intel SGX and associated costs have not been reported so far, several recent research papers (e.g., [7, 9, 17, 22, 27, 37, 43]) have shown the feasibility of side channel attacks against Intel SGX enclaves. These attacks are less costly than compromising hardware, yet technically involved and often target specific applications not hardened against them. However, it is still necessary to consider that a certain fraction of TEEs might be compromised.

Even for a potent, state-like attacker, it is not feasible to compromise the secure hardware PKI or the underlying blockchain. First of all, these attacks would compromise all solutions based on this technology, destroying core business models. As an example, the NSA might be hesitant to destroy the business model of the American company Intel by compromising Intel SGX's PKI. Furthermore, if one technology solution is compromised, SmarTor can be migrated to another system. Note that the described attacks are far more difficult than compromising three BAs not protected by secure hardware which is sufficient to compromise the current system. For an attacker with such capabilities, it is easier to write an exploit directly targeting TC software, forcing it to reveal its identity.

Adversary Goals and Capabilities Our adversary aims to influence information about Tor's networking infrastructure with the goal to, e.g., launch denial of service attacks or facilitate deanonymization of users. In particular, the adversary may introduce 'ghost' relay nodes, which do not physically exist but are listed as available nodes in the network topology information. Furthermore, the adversary may try to increase chances of malicious RN of being selected for forwarding by setting up his own BMs and reporting high bandwidth for RNs he controls and bad bandwidth or status 'unavailable' for any other RN. While operating his own BM, the adversary has full control over BM-Host and may manipulate any inputs and outputs coming to/from BM-TEE. For instance, he may try to delay networking packets to artificially decrease measured throughput, replay measurement results from previous measurement rounds, or try to manipulate reports produced by BM-TEE. The adversary may also compromise the BM-TEE on his platform but is limited in the number of BMs he fully controls.

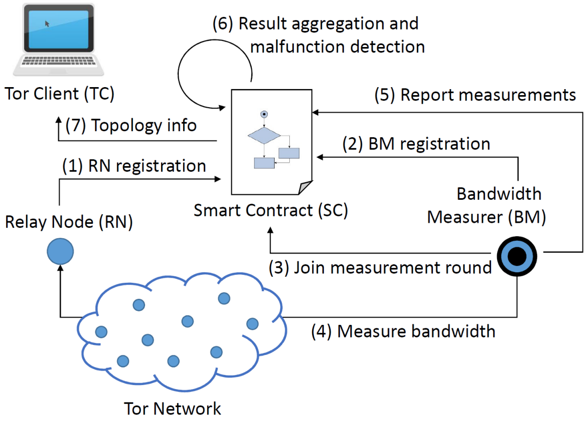

Figure 2: SmarTor Functionality

4 DESIGN

The high-level idea of SmarTor is to realize the functionality of Directory Authorities (DAs) in a Smart Contract (SC) in order to further distribute trust into core infrastructure components and change the underlying trust assumptions.

Challenges Functionality that can be implemented in smart contracts is limited. In particular, it is impossible to realize the functionality of the Bandwidth Authorities (BAs) verifying the advertised bandwidth since it would require synchronous communication over the network, precise timer, and the ability to protect cryptographic keys by the contract. Additionally, one needs to account for the fact that SC-based computations are quite costly. Hence it is a necessity to make them as simple as possible.

General Approach Our approach to solve these challenges is to outsource functionality of BAs to (untrusted) third parties, which we call *Bandwidth Measurers* (BMs). BMs perform measurements in measurement rounds orchestrated by the SC and report them to it for aggregation. Since we assume that BMs are untrusted, we introduce redundancy in the system and perform statistical analysis of reports provided by many BMs in order to detect misbehavior. This analysis allows the SC to maintain a reputation value for every BM and exclude misbehaving BMs. We furthermore introduce secure hardware which significantly raises the bar for attacks against BMs. This approach allows us to reduce redundancy and simplify computations.

System Overview We depict the high-level overview of our system in Figure 2, which illustrates involved entities and their interaction in the following use cases: (1) RN registration, (2) BM registration, (3) join measurement round, (4) bandwidth measurement, (5) report measurements, (6) result aggregation, (7) malfunction detection, and (8) topology information distribution. Below we describe each of the use cases in details. The choice of the system parameters is discussed in Section 6.

4.1 Entity Communication

As mentioned in Section 3, BM-Host is untrusted. Yet, BM-TEE lacks network access, and such can not interact with other entities (like the SC) on its own. Because of this, all communication of BM-TEE with the outside world must be mediated by BM-Host. We discuss the resulting security implications in Section 5.2.

Communication Keys Every entity has a set of keys available. Their notation and usage are shown in Figure 3. Note that *Tor session keys* are temporary and that BM-Host has no access to keys held by BM-TEE.

Message Forwarding Transactions from BM to the SC are created by BM-TEE and signed by its wallet key sec_w. BM-Host forwards these transactions to the miners which then include them into the blockchain. Similarly, BM-Host provides the current version of the blockchain to BM-TEE which will verify it. Messages between BM and RN are encrypted and authenticated with a session key sec_s. BM-Host forwards any incoming messages to BM-TEE and any outgoing messages to the specified RN.

4.2 Relay Registration

The goal of RN registration is to inform SC that a new RN wishes to join the network and to provide its contact information to Tor users. In order to do so, a new RN sends a registration transaction τ_{RN} signed by its wallet key to the SC. This transaction includes, like in the current system, RN's `access_information` like `ip`, `tor_identity_key`, and `exit_policy`. Upon reception, the SC saves this data in the list of RNs, enabling Tor users to use this relay in circuits. It further ensures that only transactions signed by the same wallet key may change this information later on.

4.3 Bandwidth Measurer Registration

Before being able to participate in measuring rounds, a BM has to register itself with the SC, as shown in Figure 4. The registering BM (BM_{new}) issues the registration transaction τ_{BM} and sends it to the SC (step 1). In step 2, one of the already registered BMs verifies the attestation data of BM_{new} to ensure that its platform has a genuine TEE. Step 2 will be repeated by every registered BM, which then submits its vote on trustworthiness τ_{vote} of BM_{new} to the SC (step 3) which will verify and count those votes (step 4). If the majority of registered BMs voted in favor of BM_{new}, the SC adds it to the list of all registered BMs (step 5).

Registration Before registration, BM-TEE of BM_{new} creates its wallet key and seals it to persistent storage. Afterwards, it creates τ_{BM} including BM_{new}'s `tor_identity_key` and linkable (cf. Section 2.2) `attestation_data` including pub_w.

Verification Process This attestation data is used to verify that sec_w was indeed created within a genuine TEE and that no other wallet key (representing a BM) was created from this TEE. Since this verification is computationally expensive, the SC only verifies the initial group of BMs (*bootstrapping phase*). Afterwards (*production phase*), this process is outsourced to the already registered BMs. Every BM-TEE should periodically query the blockchain for attestation data of a new BM and verify it. Afterwards, it will create τ_{vote} including `verification_result` and `last_blockhash` (the hash of the last block read by BM-TEE) to ensure freshness.

Voting and Adding The SC will save this result if τ_{vote} was signed by a registered BM and `last_blockhash` is recent enough. The SC furthermore calculates the distance between the blocks containing τ_{BM} and τ_{vote}. If this exceeds a pre-defined limit $dist_{vote}$, the vote counting process is triggered. If the majority of the received votes

Key name	Notation	Cryptosystem	Hold by	Usage
Wallet key	(sec_w, pub_w)	EC(secp256r1)	BM-TEEs, RNs	Sign transactions to SC
Tor identity key	(sec_t, pub_t)	RSA2048	BM-TEEs, RNs	Key exchange to create sec_c
Tor session key	(sec_c)	AES128-GCM	BM-TEEs, RNs	Protect Tor session communication
Sealing key	(sec_s)	AES256-GCM	BM-TEEs	Encrypt data to persistent storage
Attestation key	(sec_a, pub_a)	EPID	BM-TEEs	Sign attestation data

Figure 3: Available entity keys with their notation, usage, and cryptosystem in our implementation.

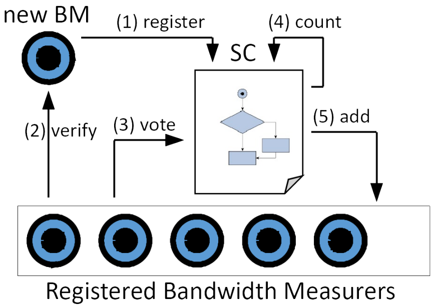

Figure 4: Overview over a successful registration process.

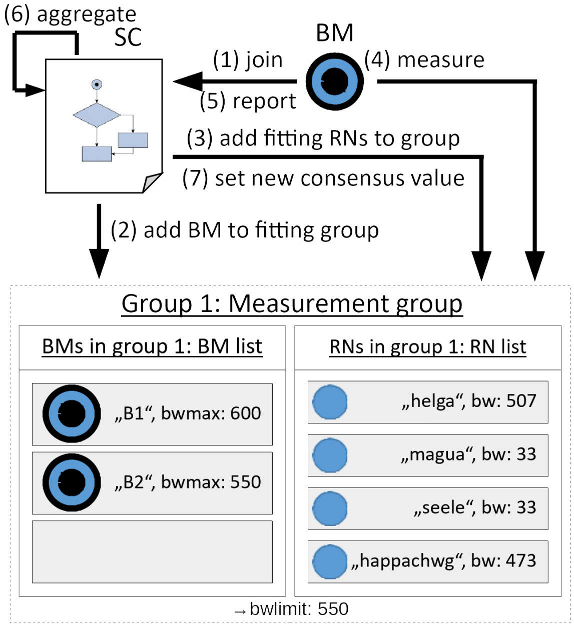

Figure 5: Overview over a successful measurement round.

were in favor of the new BM, it is added to the list of registered BMs, enabling it to participate in measurement rounds. Additional votes after this process are discarded.

4.4 Join Measurement Process

The measurement process as shown in Figure 5 is done in rounds orchestrated by the SC. Each BM has to actively join a measurement group which starts as soon as enough BMs are available. The SC creates different groups for BMs of different bandwidth and assign suitable RNs to it. Measurement results are reported to the SC, which will then aggregate them to a new consensus value for each RN.

Joining Measurement Rounds In order to join a measurement round, BM-TEE creates a joining transaction τ_{join} signed by its wallet key. This transaction includes `rnd`, `ctr`, and `bwmax` where `rnd` is a random number, `ctr` is the amount of joining transactions created by this BM and `bwmax` is the maximum bandwidth this BM can measure.

Joining Verification After receiving the transaction, the SC verifies whether this BM may participate in measuring rounds and whether *ctr* monotonically increased by one compared to the previous transaction. The last step is essential to prevent possible attempts to brute-force a specific `rnd` value. If the verification succeeds, the SC randomly [2] puts it in one of the pending measurement groups.

Relay Assignment After *n* people joined an open measurement group, the SC randomly assigns *r* relays to it. For this, the SC randomly picks up a RN out of the list and adds it to the group if this does not violate group properties. As such, every BM must be able to measure this RN. The last measured bandwidth (bw) of this RN must be lower than `bwsize` which is the lowest `bwmax` value of BM's in this group. If the RN was not measured before or the last measurement failed, bw is not set. In this situation, the RN is added to the group unless there is already a limited amount *u* of such RNs. This process is repeated until the group is full.

4.5 Bandwidth Measurements

After joining a group, a BM should query the state of the blockchain and wait for information about the assigned relays. Once they are available, it will start the measuring process. During the measurement process, the BM repeatedly fetches a file over custom circuits and produces a result transaction τ_{res} which can be forwarded to the SC.

Building Measurement Circuits Different from the current approach, measurement circuits in SmarTor consist of three RNs. The RN to be measured is the middle node, and two faster RNs are used as entry and exit. Since the RN to be measured offers the lowest bandwidth, the bandwidth of the whole circuit is equivalent to the bandwidth of this RN.

Choosing Entry and Exit Nodes BM-TEE randomly picks entry and Exit RNs. [3]. As bandwidth may vary, the entry and exit should have far more bandwidth than the RN to be measured. If no such relays are available, as it is the case for the fastest RNs, BM-TEE randomly chooses a RN out of the fastest RNs available.

[2] We will discuss the problem of getting randomness in SCs in Section 6.1.

[3] Note that BM-TEE has access to real randomness from hardware (cf. Section 6.2).

Measurement Process BM-TEE decides the order in which RNs are measured at random. After circuit creation, the BM will connect over Tor to one server out of a predefined list. It then repeatedly fetches a file over this circuit; the average bandwidth provided by the stream during this process is the measurement result.

Honeypot Measurements At random points during the measurement process, BM-TEE also measures h RNs of similar bandwidth as the other RNs. These measurements act as honeypots to detect problems with the connection and message forwarding of BM-Host (cf. Section 6.2). BM-TEE stops the measurement process if the measured bandwidth of these honeypot RNs does not align with the last measured value.

4.6 Reporting and Aggregating Results

After successfully collecting all measurements, BM-TEE creates a result transaction τ_{res}. BM-Host should forward this transaction to the SC which verifies these votes and appoints the newly assigned bandwidth values.

Result Transaction The result transaction τ_{res} contains the measurement results of all measured relays. This transaction (like every other) is atomic implying that either every result or none can be transmitted to the SC. Further information in this transaction includes a `counter` describing the number of times BM-TEE has run the measurement script, and the `blockhash` of the latest block in the blockchain read during that process.

Result Verification After receiving τ_{res}, the SC verifies that the BM may report results for this group and `blockhash` is a block after the block the group was started. It further validates that `counter` monotonically increased by one to ensure that BM-Host can not influence the bandwidth by starting the measurement process multiple times. If no further results are missing or the distance between the current block and the initiation of the group exceeds a predefined limit $dist_{res}$, the SC will aggregate the results.

Result Aggregation If more than half of the assigned BMs failed to report results, the results of the measurement round are discarded. Otherwise, like in the current aggregation method, each measured RN is appointed the (lower) median value of its measurement results as the new bandwidth consensus value.

4.7 Malfunction Detection

We assumed (cf. Section 3) that a certain fraction of TEEs can be compromised. As a countermeasure, we want to exclude the measurement results of misbehaving BMs in aggregation and force them to undergo registration again. During aggregation, the SC calculates a *reputation score* r_{BM} for each BM which is used as a measure for trustworthiness. The lower the reputation score, the higher the chance for a BM to be excluded from the list of registered BMs.

Reputation System After aggregation, every BM of the group gets an *Activity Point* (AP) for participating in this measurement round. It further gets an additional *Reputation Point* (RP) if its reported measurement is aligned to the consensus value. The ratio of RP to AP can be seen as an indicator of how similar a BM behaves compared to the other BMs and is used as r_{BM}.

BM Exclusion After reporting a bandwidth result the SC has a probability of excluding the BM out of the group of registered BMs.

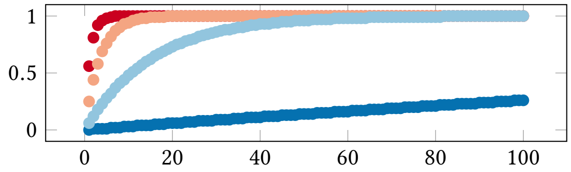

Figure 6: Probability of exclusion (y-Axis) after participating in multiple measurement rounds (x-Axis) for —●— r_{BM} = 25%, —●— r_{BM} = 50%, —●— r_{BM} = 75%, and —●— r_{BM} = 95%

Parameter	Notation	Proposed Value
BMs per group	n	9
RNs per group	r	15
max. new RNs per group	u	2
Honeypots per round	h	2
Voting timeout	$dist_{vote}$	256
Reporting timeout	$dist_{res}$	256

Figure 7: Used parameter and proposed values.

This is done by randomly picking [4] a number $rnd \in [0, 999] \subset \mathbb{Z}$. The BM is excluded if $rnd \leq f(r_{\text{BM}})$, where $f : [0, 1] \subset \mathbb{R} \rightarrow [0, 1000] \subset \mathbb{N}$ is a monotonic function used to smooth the probability curve for exclusion. Note that the value range of f implies that any BM has a 0.1% chance of exclusion regardless of its reputation score. This ensures every BM has to undergo re-validation from the others at some point. If the TEE is known to be compromised, its keys are revoked, such preventing rejoining during the verification step at registration.

Function Choice We propose to use $f(r_{\text{BM}}) := \lfloor (1 - r_{\text{BM}})^2 \cdot 1000 \rfloor$ as exclusion function. Probabilities for exclusion after multiple measurement rounds are shown in Figure 6. While a BM with a reputation score of 50% has a 90% probability of exclusion after 8 rounds, an honest BM (r_{BM} = 95%) only has a 26% chance of exclusion after 100 rounds. Note that this function can be implemented using basic operations and r_{BM} = 100% is unlikely as bandwidth differs over time.

5 SECURITY ANALYSIS

The main security goal of our solution is to hinder attacker's ability to artificially inflate their bandwidth without providing the proportional amount of resources. In the following section, we will discuss potential attack scenarios and their influence on SmarTor. We specify the system parameter introduced in Section 4 in Figure 7.

5.1 Group Compromise

As BMs get randomly allocated to a certain group, there is a nonzero probability $gc \in [0, 1] \subset \mathbb{R}$ that an attacker controlling many BMs gets assigned enough of them into the same group to represent the majority in this group. We will call such a group *compromised* and denote $p \in [0, 1] \subset \mathbb{R}$ for the percentage of entities under the attacker's control. As each BM has the same probability to be chosen for a certain group, gc follows a binomial distribution with $gc = \sum_{\lceil i=n/2 \rceil}^{i=n} \binom{n}{i} p^i q^j$, $q := 1 - p$, and $j := n - i$. Since $gc < p$, the

[4] See Section 6.1 for Randomness in SCs.

n	9	9	9	29	9	49	99
p in %	5	10	25	25	40	40	40
gc in %	0.00	0.09	4.89	0.18	26.66	7.76	2.19

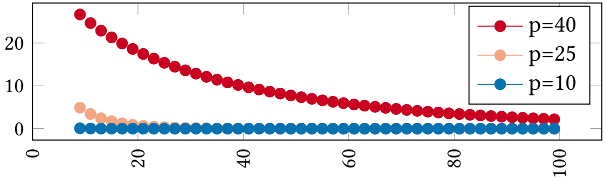

Figure 8: Probability gc (y-Axis) in regard of p and n (x-Axis).

majority of the groups are not compromised as long as an attacker does not control the majority of entities.

In Table 8, we show the probabilities to compromise a certain group for different p and n. For instance, for $p = 10\%$ and $n = 9$, the attack success probability is 0.09%, which is small. Even an attacker with $p = 40\%$ only has a 26.66% chance to overtake a group if using $n = 9$. However, this would mean that an attacker would need to compromise $120 = 40\% \cdot 300$ systems if assuming that, like in the current system, there are around 6000 RNs and only 5% of them participate as a BM.

Consequences Compromise of a group is unlikely as gc is generally low for most attackers and is further reduced if raising either n or the amount of BMs. It furthermore is more difficult to compromise a group than compromising three BAs. The possible influence of an attacker who compromised a group depends on whether he controls BM-Host or also compromised BM-TEE.

5.2 Attacks from a malicious Host

Here we discuss attack scenarios where an attacker can compromise hosts of BMs he controls, but not their BM-TEEs. Since BM-Hosts mediate communication between BM-TEEs and the SC, they can delay or drop forwarded packets and transaction.

Delaying Packets during Measurements BM-Host can monitor the bandwidth usage of the system and identify measurements. Hence, an attacker might choose to delay forwarded packages, thus reducing the measured bandwidth. However, since BM-TEE decides the order of measurements and every BM in a group has a similar bandwidth, an attacker does not know which BM is measured right now. It might still be beneficial for an attacker to delay packets during the whole measurement process, for instance, if he does not control any of the assigned RNs. However, BM-TEE can detect this attack by observing the discrepancy between expected and measured bandwidth for honeypot RNs (cf. Section 4.7). In this case, it will not create the result transaction.

If using the parameters n = 9, r = 15, h = 2 and randomly delaying the measurements of one RN in each measurement, an attacker has only an 11.92% chance of being captured by honeypot measurements. Still, this does not influence the consensus unless he did so on at least five BMs in the measurement group. Even then, he has to delay the same RN at least five times to influence the consensus. Choosing a RN to delay randomly, this has the small probability of 0.01%. If he fails to influence the consensus, his attack attempt leads to a decrease in the reputation value of all BMs involved in the manipulation. Raising either n, r, h, or the amount of BMs further lowers the success probability of such an attack.

Dropping Reports An attacker controlling BM-Host might have no access to or decide to drop the result transaction τ_{res} instead of forwarding it to the SC. In this case, its BM is assigned no reputation pointss, decreasing its reputation score. This eventually leads to the SC excluding it from the group of registered BMs. As the attacker did not compromise BM-TEE in this scenario, the BM may rejoin again after a certain amount of time. Yet, this does not reset its reputation score. Similar behavior leads to a further decrease in its reputation score until SC eventually excludes BM after every measurement. Given the fact that an attacker has to pay for gas for re-joining, following this attack strategy may impose significant costs.

If the attacker is also able to compromise the current group, he can prevent the building of new consensus values as this is only done if the majority of BMs reported a result [5]. However, in order to launch a DoS attack and prevent a RN from being measured (again), an attacker would have to compromise every group containing this RN which is even less likely [6] to happen consistently.

5.3 Attacks from compromised TEEs

While compromising secure hardware is costly and requires non-trivial skills, a motivated attacker might be tempted to exercise this attack vector, if the attack revenue outweighs invested efforts. Hence, we elaborate on scenarios where an attacker compromises BM-TEEs, and discuss security implications.

Compromising a few BM-TEEs In a first scenario, an attacker has some BM-TEEs under his control, but he did not compromise the group. Such a BM can report arbitrary measurements and, for example, assign high bandwidth to attacker-controlled RNs and low bandwidth otherwise. This attack, however, is detected during the aggregation process, since the SC excludes outlier values during consensus aggregation. Furthermore, such attempts reduce the reputation score of the attacker-controlled BM leading to exclusion of the BM from the group of registered BMs, forcing it to undergo the re-joining process. As before, even if compromised TEE attestation keys are (not yet) revoked, re-registering imposes an additional financial burden on the attacker. As long as an attacker cannot compromise a group, the SC ignores all measurement data of compromised BMs and decreases their reputation score.

Compromising many BM-TEEs In our second scenario, an attacker has compromised enough BM-TEEs to compromise a measurement group. For such groups, the attacker can influence the assigned bandwidth value of all RNs. Furthermore, his reputation scores are increased, while reputation scores of the honest BMs are decreased. Hence, there is a strong necessity that this scenario is as unlikely as possible. However, this attack is far more sophisticated than compromising the majority (right now: three) of BAs in the current system. Furthermore, its influence is limited, as non-compromised groups still produce valid measurement results and increase the reputation score of honest BMs. Lastly, under our assumptions, the majority of groups is never compromised as one

[5] This prevents an attacker from influencing the consensus by being the only reporting entity if launching a network DoS attack against other BMs.

[6] The probability to compromise i groups is gc^i.

can expect $gc \cdot groupcount$ compromised groups and $gc < p < 0.5$. Hence, we consider SmarTor to be a significant improvement over the current system.

5.4 Attacks on the SC

Information Flooding An attacker might try to perform a DoS attack on SmarTor by registering *ghost relays* which do not exist. Since the bandwidth value is set to 0 before the first measurement takes place, such an attack would not affect Tor users. Furthermore, an attacker will have to pay gas for adding each RN to the blockchain, which imposes enormous costs. At the same time, an increased amount of RN data stored in the blockchain will increase execution cost of the SC, since the SC needs to create more random numbers until it finds enough "old" RNs to put into a group. One solution to prevent such an attack is to keep relays whose last measurement was not successful (or existing) in a separate list and add them to the list of verified RNs after a valid measurement. In this case, a constant amount of RNs from the unmeasured list is added to each group, ensuring that every newly registered RN is measured at some point. A deposit for registration, which is only paid back upon successful registration, can be introduced to prevent a similar attack against the BM registration system.

Relay Selection The SC only adds RNs with suitable `bwsize` to a group. In order to limit the amount of RNs the SC can choose, a BM could report unreasonable low `bwmax`. Introducing a bandwidth threshold for BMs prevents such an attack.

6 IMPLEMENTATION

Our prototype consists of three major components: (i) Smart Contract, (ii) the measurement script that runs within Intel SGX, and (iii) the host of the BM, leading to a total of 4483 Lines of Code (LoC). For compatibility reasons, communication keys are instantiated using the underlying cryptosystem of the currently deployed systems.

Parameter choices Figure 7 shows an overview of the parameters in SmarTor. We propose the example values from Section 5 as they offer reasonable security and a further increase in n or r would also increase the system cost. The choice for the timeouts is driven by the fact that the function `blockhash` of the Solidity API is only able to resolve the blockhash of the last 256 blocks. We furthermore interpret a measurement as valid if the result is within ±30% of the expected value.

6.1 Smart Contract

Our proof of concept implementation of the SC was developed using Ethereum platform and consists of 353 LoC in the Solidity 0.4.20 language.

Implementation It saves information associated with BMs, measurement groups, and RNs. We present a full system model showing the stored information, used structures, and possible interaction in Appendix C. All variables which are not explicitly set with a parameter of a specific transaction default to 0. To the variables mentioned in Section 4, the SC saves a `state` variable for each BM. The current state of a BM limits the allowed interaction with the SC, so that, e.g., a BM which did not register may not vote on new BMs. The states and transition events are shown in Figure 9 and explained in more Detail in Appendix C.

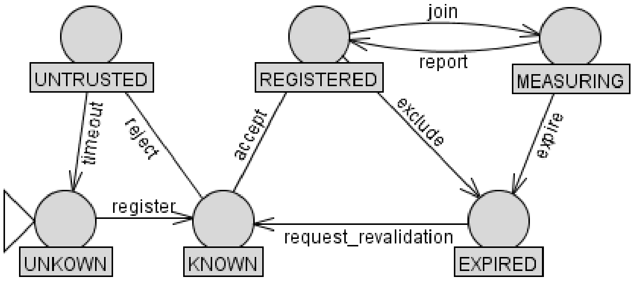

Figure 9: BM states and transition events.

SC Attestation As of right now, SGX's attestation data has to be verified by the *Intel Attestation Service* (IAS, cf. Appendix A). While this design choice ensures that verification and revocation checks are handled correctly, it also requires a synchronous communication channel not available inside the SC. As a workaround, we suggest that the nine current DAs take the role of the initial BM group, as they are already trusted in the current system. Note that trust into this initial group is limited to the bootstrapping phase as more BMs join and verify attestation data.

Limitations While Smart Contracts languages are Turing-complete in theory, there are several simple operations which require complex workarounds in practice. This is also true for Solidity which, to the best of our knowledge, is the most potent SC language. Due to the *gas limit* for SC execution, the possible scope of workarounds is restricted as this limit cannot be raised arbitrarily [7]. In practice, the following functionality should be introduced to Solidity before implementing a full version of SmarTor: (1) Fixed-point numbers and computations, (2) Iteration over key/value sets of maps, (3) Delete maps, (4) Return block number for block hash, (5) Pseudo-Random Number Generators (PRNGs). Due to these limitations, our Proof-of-Concept implementation only provides full functionality for one measurement group. However, this is sufficient to estimate the gas cost of such a system.

Randomness SCs have no native access to randomness. How to create randomness in SCs is an orthogonal problem, which is explored separately [24, 25] and not specific to our application. It is possible to use pseudo-randomness (e.g., from the block headers) but that mechanism must be hardened against a possible influence of the miners [6]. In our implementation, the SC selects RNs for a measurement group by using real hardware randomness provided by BM-TEEs during joining of the group. The SC aggregates it at the start of the group by hashing the concatenated random values. This hash value then seeds a PRNG [8]. Since this randomness is only available once a group started, this approach cannot be used to assign BMs to groups, and other approaches have to be explored if extending the implementation to multiple groups.

[7] This prevents a DoS attack on miners where a SC function takes longer to execute than it takes other miners to create a new block.

[8] As there is no implementation of a PRNG in Solidity yet, we use $sha3(prev\_rnd)$ mod max as approximation to create a random number $rnd \in [0, max - 1] \subset \mathbb{N}$.

6.2 Bandwidth Measurement Script

We implemented bandwidth measurement script as an Intel SGX enclave. Our implementation is written in C++ using the Intel SGX SDK 1.9 and consists of 3018 LoC. The resulting library offers ECalls to trigger functionality for registration (`create_wallet`), joining measurement rounds (`join_group`), reporting measurement results (`measure_bandwidth`), and expiring oneself (`expire`). Every ECall results in a raw transaction signed by the wallet key created during `create_wallet`. As shown in Figure 3, IntelSGX and Ethereum use different curves. Some cryptographic functionality was rewritten [9] using the mbedtls library for SGX [34] and the Portable C++ Hashing Library [8]. This signed transaction is saved to a file from where BM-Host can forward it to the SC. For example, `create_wallet` results in the registering transaction τ_{BM} which in our implementation can also be used to request re-joining. If an ECall fails (e.g., when trying to expire before creating a wallet), the enclave will throw an exception, terminating without producing a transaction.

We developed our system using pre-release mode which, contrary to release mode, uses unsigned enclaves and such cannot produce attestation statements. In practice, the distributor of the program code (i.e., the Tor Foundation) would have to sign every published enclave before deployment with an *enclave signing key* requested from Intel.

We use the function `sgx_read_rand` to create truly random numbers (e.g., for key creation), `sgx_create_monotonic_counter` for transaction counters, and `sgx_get_trusted_time` as the time source. While the resolution of this time source is limited to seconds, we were still able to achieve precise measurement results (cf. Section 7.1).

6.3 Bandwidth Measurement Host

We used usual Tor and Geth clients for Windows for the host to support communication with Tor and Ethereum networks, respectively. Our host implementation consists of 1112 LoC which is responsible for forwarding transactions to the SC, starting the enclave, and fulfilling requested OCalls.

The OCalls necessary for networking are implemented using the `winsock2` system library for Windows. An additional script is checking whether the enclave saved any new transaction to a folder. If so, it forwards it to the Ethereum network using the `sendRawTransaction` function of EthereumJS [33].

Furthermore, a Tor Client should offer its Socks 5 proxy [23], used to forward messages to another relay, on port 9050, and a control port on 9051. BM-TEE uses the control port to create circuits and perform reconfiguration necessary for the measurement process. In particular, circuits are not created automatically, but rather by control commands from the enclave. Note, that generally, communication to Tor and the Tor keys need to be protected by Intel SGX as well. This can be achieved by using a secure Tor implementation like SGX-Tor [21] [10].

7 EVALUATION

In this section, we provide the results of our system evaluation. We evaluate our bandwidth measurements with regards to precision and reliability, and compare it to TorFlow [30, 32], the solution currently used by Tor. Furthermore, we evaluate our smart contract with respect to the gas cost necessary for its execution.

7.1 Measurement script

In order to obtain measurement results similar to ones from real-world deployment, we ran our measurement script using Intel SGX hardware and measure the bandwidth of real Tor RNs that are not under our control. We used a Lenovo T470s with an Intel i7-7500U processor (with SGX) running a 64-bit Windows 10 Education as an operating system. The evaluation was taking place on 14 full days between 2018-05-18 and 2018-05-31 [11].

Methodology We built measurement circuits by placing the RN to be measured in the middle of the circuit and choosing entry and exit RNs randomly from predefined lists we built [12]. If a measurement circuit was not build within 15s, we re-tried it up to 3 times with different entry and exit RNs. Measurements were repeated every 30 minutes. A single measurement was scheduled to repeatedly fetch a pre-defined file until at least 25s have passed to ensure the time resolution of SGX would not influence the measurement result. As our endpoint only had an upload of up to 2MiB/s, we chose eleven RNs [13] with a bandwidth of less than 1MB/s [14] for measurements to ensure that unreliability of the endpoint would not influence the measurements.

Comparison Sets We compare the results of the SmarTor measurement script (which we denote as *SMS*) with the BA bandwidth votes for the same RN in the same period [15]. As described in Appendix B, measurement values from the DA votes are commonly reused without a new measurement. Hence, we will compare our measurements to two sets of data: $DA_\bullet$ and DA_+, where DA_+ is the set that ignores repeated values, and $DA_\bullet$ includes them (though, we identified that both sets behave quite similar).

Statistical Approach To estimate precision, we use the relative difference between the median value of the measurements for one RN (med_{RN}) and the advertised value of this RN (adv_{RN}) as precision error $pe := \frac{|adv_{\text{RN}} - med_{\text{RN}}|}{adv_{\text{RN}}}$. For reliability, we used the relative interquartile range of this RN as scattering error $se := \frac{IQR_{\text{RN}}}{adv_{\text{RN}}}$.

Example In Figure 10, we depicted statistical information about the measurement results of the RN "greedygertie". This relay advertises a bandwidth of $100KiB/s$ ($99.7KB/s$). Our measurements had a lower precision error $pe = 3.74\%$ than DA_+ ($pe = 70.4\%$). We furthermore have a lower scattering error ($se = 5.12\%$) than DA_+ ($se = 13.31\%$).

[9] While we were grateful that we were offered to reuse code from the TownCrier project [47], we opted not use it as their license is incompatible with open source projects like Tor.

[10] While we are using the most recent version of Intel SGX SDK (1.9), SGX-Tor is implemented for an older, incompatible, version (1.6). This is a temporary limitation until a newer version of SGX-Tor becomes available.

[11] Unless denoted otherwise, all point of times are given in ISO-8601 for UTC.

[12] In particular, we have chosen "0x3d004", "0x3d005", "Lule", "BlockHouse2", "Rudeboy", "CatRelay" as potential entries, and "xanadurregio", "DigiGesTor1e1", "ins0", "noiseexit03d", "CalyxInstitute15", "IPredator" as potential exits.

[13] We chose the following RNs (bandwidth in KiB/s): "seele" (100), "greedygertie" (100), "gnarzkorf2" (100), "kwail" (100), "tiger" (250), "helga" (400), "Ograoum" (900), "KAMIKAZE" (1000), "kolben" (1000), "panic" (1000), "lkjhgfdsa" (1000).

[14] As of 2018-06-15, 40% of RNs advertise a bandwidth value in this range. The median advertised bandwidth value is 1.8MB/s, the upper quartile has between 7.2 and 9.7MB/s.

[15] These votes are archived at https://collector.torproject.org/archive/.

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
SMS	975	12	91	94	91.1	96	99
DA$_+$	95	15	25	29	30.8	38	47

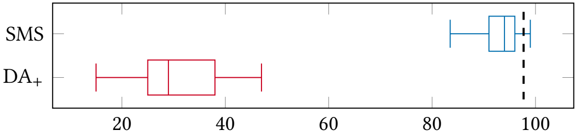

Figure 10: Measurement results for "greedygertie" ($97.7KB/s$) as table and boxplot in KB/s. Outlier points are not shown.

Method	RN set	med(pe)	sum(pe)	med(se)	sum(se)
SMS	RNs$_+$	4.77%	107.25%	5.12%	68.30%
DA$_+$	RNs$_+$	36.51%	283.67%	14.34%	141.57$
SMS	RNs$_\bullet$	44.60%	769.38%	19.97%	624.76%
DA$_+$	RNs$_\bullet$	54.62%	552.20%	35.53%	403.39%

Figure 11: Precision and scattering of the measurements.

action	gas / 1000	US-$ (ETH)	US-$ (ETC)
SC Deployment	3851	5.777	0.173
RN Registration	112	0.168	0.005
BM Registration	930	1.395	0.042
BM joining	101	0.152	0.005
BM voting	88	0.132	0.004
RN assignment	375	0.562	0.017
BM reporting	255	0.383	0.011
group aggregation	426	0.639	0.019
BM expiring	28	0.042	0.001

Figure 12: System cost for SC events in gas and US-Dollar.

Outlier RNs From the eleven RNs we measured, five did not produce expected results, i.e., neither DA$_+$ nor *SMS* produced a value within ±30% of adv_{RN}. Interestingly, we underestimate the bandwidth values while the current measurement method overestimates them. This effect probably is a consequence of the ability to temporarily burst the current bandwidth of a Tor RN. This ability has a higher influence on the current measurement script as their measurement duration is shorter. We write RNs$_\bullet$ for the set of all RNs and exclude the outliers in RN$_+$.

Results Figure 11 shows the sum and median of precision and scattering errors. The median *se* for *SMS* is only half as big as the one for *RN*$_+$, while *RN*$_+$ has a seven times higher precision error. The grouping process of the current speedracer measurement script prevents that RNs get assigned a bandwidth value which is too far from the advertised value. This leads to a higher precision and lower scattering for outlier RNs in DA$_+$. We list the detailled results of all RNs (also for the set DA$_\bullet$) in Appendix D.

Summary We successfully measured eleven RNs over a period of two weeks. Five of these RNs did not produce expected values with either method. For the other RNs, our measurements were seven times as precise and two times as reliable as the current method.

7.2 Smart Contract

We evaluated our SC with regards to the gas needed to call certain functions. As shown in Figure 12, deploying the SC and registering as a BM are the most expensive actions. However, deployment only has to be done once and registering/re-joining are not executed frequently. The high gas cost of the BM registration is caused by the attestation data which needs to be saved. For Intel SGX, this data takes up at least 1116 bytes.

To estimate the real world cost of our system, we assumed a gas price of 3 gwei and an exchange rate of 500$ for one Ether (ETH), respectively 15$ for one Ether on the Ethereum Classic (ETC) network [16]. Due to the much higher value of ETH, it might be preferable to deploy the system on the latter one. This is reasonable as an attacker still would need to create more than 3TH/s in order to gain 30% of the underlying hashing power necessary to perform some mining attacks [16, 29]. Assuming a power consumption of 6W for 1MH/s, this would still require 18MW, equivalent to the production of a small nuclear power plant.

With these considerations, a full measurement round [17] would cost 0.18$ for 15 relays. If like in the current system (cf. Appendix B), every relay is measured around 60 times a week, this would result in a system cost of less than 3$ per relay and month.

8 RELATED WORK

In this section, we overview works related to bandwidth balancing in Tor, discuss proposals to leverage Intel SGX for improved security, and review papers that investigated Tor and blockchain integration.

Bandwidth balancing in Tor The shortcomings of the currently used speedracer measurement system in Tor lead to the *EigenSpeed* project [39, 40], which was improved in the *PeerFlow* project [18]. PeerFlow assumes that a certain fraction of RNs is *trusted*. However, this limits scalability as a bigger network requires more trusted RNs and additional trust assumptions hinder the initial design goals of Tor. Different from PeerFlow, SmarTor does not collect meta-data about real traffic in Tor. Furthermore, our trust assumptions are reduced significantly: Neither software of BMs, nor users that operate them, nor the SC is trusted in SmarTor.

SGX-supported applications Intel SGX has received much attention recently and was leveraged to enhance security and privacy of platform architectures, systems, and applications. However, applications must be ported to SGX. Frameworks to run unchanged binaries are presented by [4] for Windows and [42] for Linux. Other frameworks aim to improve security in distributed and cloud computing [12, 15, 36, 46].

The most relevant to our work is the *SGX-Tor* project [21] that leverages Intel SGX for hardening Tor's client software. Generally, Tor's adversary model assumes that the client's software is trusted. Otherwise, the client could send the host's IP address or cryptographic keys to an attacker. SGX-Tor eliminates these attack vectors by excluding the client's software from the trust assumptions. While generally, SGX-Tor aims to solve an orthogonal problem, it can complement our system in order to enhance the security of Tor clients. Furthermore, we similarly use Intel SGX to eliminate trust in software running by BMs.

[16] These assumption are based on the data published at bitinfocharts.com.
[17] Consisting of 9 BMs joining, RN assignment, 9 BMs reporting, and aggregation.

Tor and blockchains Synergy between Tor and blockchains was investigated in several previous works. In particular, Biryukov and Pustogarov [5] propose to use a proof-of-work based blockchain for incentivizing Tor RNs. In their system, they introduce priority tickets, which can be obtained by Tor clients if they solve a Proof-of-Work (PoW) cryptographic puzzles published by Tor RNs.

The Onion Name System (ONS) [44] proposes a DNS service for Tor, which makes addresses of hidden services more memorable. It relies on a randomly selected subset of RNs to host a DNS system. To mitigate against malicious domain registration attacks, such as land-rushing and flooding, ONS introduces a PoW based lottery system, which enforces some cost for the registration of a new domain name. While ONS uses PoW blockchain to mitigate various denial-of-service attacks, we utilize blockchains as a means to decentralize trust and reduce trust assumptions.

9 CONCLUSION AND FUTURE WORK

In this paper, we presented SmarTor, an approach to increase the resilience of the Tor anonymity network against attacks on core infrastructure components and targeting manipulation of topology information. SmarTor shifts trust from Directory and Bandwith Authorities operated by a small number of individuals to a blockchain, which is much harder to compromise, since its security is backed up by a computing power of millions of miners around the globe. Furthermore, SmarTor relies on modern secure hardware technology, honeypot measurements, and a reputation system to protect the bandwidth measurement process from manipulation. We provided a proof-of-concept implementation and showed that our solution produces more reliable and valid bandwidth measurements than the current system. In SmarTor, we assume that bandwidth measurers are operated by volunteers who, similarly to Tor relays, are not getting any rewards for their service. However, since interactions with the smart contract impose costs for gas, we plan to investigate additional incentive mechanisms for bandwidth measurers in our future work.

REFERENCES

[1] Tiago Alves and Don Felton. 2004. TrustZone: Integrated Hardware and Software Security. *Information Quaterly* 3, 4 (2004).

[2] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Vol. 13.

[3] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. 2007. Low-resource Routing Attacks Against Tor. In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society (WPES '07)*. ACM, New York, NY, USA, 11–20.

[4] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding Applications from an Untrusted Cloud with Haven. *ACM Trans. Comput. Syst.* 33, 3 (Aug. 2015).

[5] Alex Biryukov and Ivan Pustogarov. 2015. Proof-of-work as anonymous micropayment: Rewarding a Tor relay. In *International Conference on Financial Cryptography and Data Security*. Springer, 445–455.

[6] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. 2015. On Bitcoin as a public randomness source. *IACR Cryptology ePrint Archive* 2015 (2015), 1015.

[7] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software Grand Exposure: SGX Cache Attacks Are Practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. USENIX Association, Vancouver, BC.

[8] Stephan Brumme. [n. d.]. mbedtls-SGX. http://create.stephan-brumme.com/hash-library/.

[9] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2018. SgxPectre Attacks: Leaking Enclave Secrets via Speculative Execution. *arXiv preprint arXiv:1802.09085v1* (2018).

[10] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86.

[11] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal hardware extensions for strong software isolation. In *USENIX Security Symposium*.

[12] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan. 2008). https://doi.org/10.1145/1327452.1327492

[13] Maxence Delong, Eric Filiol, Clément Coddet, Olivier Fatou, and Clément Suhard. 2018. Technical and OSINT Analysis of the TOR Foundation. In *ICCWS 2018 13th International Conference on Cyber Warfare and Security*. Academic Conferences and publishing limited, 164.

[14] Roger Dingledine. 2013. The lifecycle of a new relay. https://blog.torproject.org/lifecycle-new-relay. [Online; accessed 15-January-2018].

[15] Tien Tuan Anh Dinh, Prateek Saxena, Ee-Chien Chang, Beng Chin Ooi, and Chunwang Zhang. 2015. M2R: Enabling Stronger Privacy in Mapreduce Computation. In *USENIX Security Symposium*.

[16] Ittay Eyal and Emin Gün Sirer. 2014. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*. Springer, 436–454.

[17] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache Attacks on Intel SGX. In *European Workshop on Systems Security (EuroSec)*.

[18] Aaron Johnson, Rob Jansen, Nicholas Hopper, Aaron Segal, and Paul Syverson. 2017. PeerFlow: Secure Load Balancing in Tor. *Proceedings on Privacy Enhancing Technologies* 2017, 2 (2017), 74–94.

[19] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. 2013. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*. http://doi.acm.org/10.1145/2508859.2516651

[20] Simon Johnson, Vincent Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. 2016. Intel software guard extensions: EPID provisioning and attestation services. *ser. Intel Corporation* (2016).

[21] Seongmin Kim, Juhyeng Han, Jaehyeong Ha Taesoo Kim, and Dongsu Han. 2017. Enhancing Security and Privacy of Tor's Ecosystem by using Trusted Execution Environments. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*.

[22] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing. In *26th USENIX Security Symposium (USENIX Security)*.

[23] Marcus Leech. 1996. SOCKS protocol version 5. (1996).

[24] Peter Mell, John Kelsey, and James Shook. 2017. Cryptocurrency Smart Contracts for Distributed Consensus of Public Randomness. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, 410–425.

[25] Peter Mell, John Kelsey, and James Shook. 2017. Cryptocurrency Smart Contracts for Distributed Consensus of Public Randomness. In *Stabilization, Safety, and Security of Distributed Systems*, Paul Spirakis and Philippas Tsigas (Eds.). Springer International Publishing, Cham, 410–425.

[26] Chris Mills. 2016. Judge Confirms Carnegie Mellon Hacked Tor and Provided Info to FBI. http://gizmodo.com/judge-confirms-carnegie-mellon-hacked-tor-and-provided-1761191933.

[27] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. *CacheZoom: How SGX Amplifies The Power of Cache Attacks*. Technical Report. arXiv:1703.06986 [cs.CR]. https://arxiv.org/abs/1703.06986.

[28] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system.

[29] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. 2016. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 305–320.

[30] Mike Perry. 2009. Torflow: Tor network analysis. *Proc. 2nd HotPETs* (2009), 1–14.

[31] Kevin Poulsen. 2013. FBI Admits It Controlled TOR Servers Behind Mass Malware Attack. https://www.wired.com/2013/09/freedom-hosting-fbi.

[32] "TOR Flow Project". [n. d.]. Bandwidth scanner spec. https://gitweb.torproject.org/torflow.git/blob_plain/HEAD:/NetworkScanners/BwAuthority/README.spec.txt.

[33] GitHub Repository. [n. d.]. EthereumJS. https://github.com/ethereumjs/ethereumjs-tx. Accessed: 2018-06-15.

[34] GitHub Repository. [n. d.]. mbedtls-SGX. https://github.com/bl4ck5un/mbedtls-SGX. Accessed: 2018-06-15.

[35] Bruce Schneier. 2013. Attacking Tor: how the NSA targets users' online anonymity. https://www.theguardian.com/world/2013/oct/04/tor-attacks-nsa-users-online-anonymity.

[36] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. 2015. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *IEEE Symposium on Security and Privacy*.

[37] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*.

[38] Sergei Skorobogatov. 2012. Chapter 7: Physical Attacks and Tamper Resistance. In *Introduction to Hardware Security and Trust*. Springer New York, 143–173.

[39] Robin Snader and Nikita Borisov. 2009. EigenSpeed: Secure Peer-to-peer Bandwidth Evaluation. In *Proceedings of the 8th International Conference on Peer-to-peer Systems (IPTPS'09)*. USENIX Association, Berkeley, CA, USA, 9–9.

[40] Robin Snader and Nikita Borisov. 2011. Improving Security and Performance in the Tor Network Through Tunable Path Selection. *IEEE Trans. Dependable Secur. Comput.* 8, 5 (Sept. 2011), 728–741.

[41] Yogesh Swami. 2017. Intel SGX Remote Attestation is not sufficient. (2017).

[42] Chia-Che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A practical library OS for unmodified applications on SGX. In *Proceedings of the 2017 USENIX Annual Technical Conference.*

[43] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium. USENIX Association.*

[44] Jesse Victors, Ming Li, and Xinwen Fu. 2017. The Onion Name System. *Proceedings on Privacy Enhancing Technologies* 2017, 1 (2017), 21–41.

[45] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151 (2014).

[46] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. 2009. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In *IEEE Symposium on Security and Privacy.* https://doi.org/10.1109/SP.2009.25

[47] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 270–282.

A INTEL SGX DETAILS

Intel Software Guard Extensions (SGX) [10] are a special set of processor instructions which enable execution of security-critical application code on platforms controlled by untrusted parties. In particular, SGX was designed to ensure confidentiality of processed data and integrity of execution without the need to trust in the owner of the hardware or processes (including the OS) running on it. Systems enabling this security features are more generally called *Trusted Execution Environments* (TEEs).

Enclaves SGX enables process execution of programs in a protected and encrypted memory space known as an *enclave*. SGX assumes the CPU itself to be the only trustworthy hardware component of the system, i.e., enclave data is handled in plain-text only inside the CPU. SGX enabled processors are equipped with a set of hardware-protected keys. These are generated by a trusted *Platform Service Enclave* (PSE, provided by Intel) from a platform-specific secret provided by hardware and cannot be accessed by a usual enclave. The vast majority of secrets are unknown as it is assumed that it is very expensive to get knowledge of it. Intel also provides revocation information for platforms which are known to be compromised.

Attestation [2] enables enclaves to prove to a verifier that they are running inside a genuine SGX hardware. At startup time of the enclave, a hash of its initial state (*measurement*) is saved to storage protected by the PSE. An enclave may request a *report* from the PSE containing the measurement, further attributes of the enclave and custom data like a public key generated by the enclave (*user key*). A trusted *quoting enclave*, also provided by Intel, can verify whether this report indeed belongs to an enclave hosted on this platform. If it does, it will sign this report with an *attestation key* given from the PSE using EPID group signatures [20] resulting in a *quote* of the enclave. Making these quotes *linkable* to the hardware is possible. Linkable quotes enable knowledge of whether different quotes were signed on the same platform without revealing the identity of the hardware. Such a quote can be sent to other services enabling *remote attestation.* The quote can then be verified with an EPID public key certificate or by contacting the *Intel Attestation Service* (IAS). Since Intel decided to encrypt the signature of the quote [41] with a key known only to Intel, only the latter option is available at the moment.

Sealing describes the process of encrypting data to persistent storage with a key that can only be recovered by one entity. The PSW provides the *sealing key* used for this which can be based on the *enclave identity* or the *signing identity.* If using the enclave identity, only instances of that specific enclave can recover that key while access from enclaves with other measurements (including different versions of the same software) is prevented. If using the signing identity, the data can be accessed by every program created and signed by a single authority. This enables data transfer between multiple enclave programs of the same software vendor.

OCalls enable enclaves to access or write data from/to the outside world. They are used for writing (signed) results of computations to the user or performing (encrypted and authenticated) communication with other entities. Since the OS is untrusted, functions leading to a system interrupt can only by called during an OCall. Because of this, many functions (like the system time) and most of the usual C++ libraries like Boost are not available inside enclaves. However, some of the functionality is provided by the secure hardware. This most importantly includes a trusted source of time, resolving to seconds, and a trusted counter for verifying freshness against replay attacks with sealed data.

ECalls enable the calling of function inside enclaves. They mark the allowed entry points for code execution. Once called, the enclave will execute commands until terminating the requested execution either by reaching the end of the source code or the host shutting down the enclave. OCalls and ECalls mark the interface of the enclave to the rest of the system. They are defined using the *Enclave Definition Language* (EDL).

B TOR SPEEDRACER MEASUREMENTS

In this section, we give an overview of the topology distribution process used by the DAs. Furthermore, we briefly analyze results of the measurement script as published in the measured bandwidth values in the votes of the BAs, look at the amount and quality of measurements, and discuss the measurement method. Every date format is given in the ISO 8601 standard and referring to the "valid-after" tag of the underlying vote or consensus. All measurement results are given in KB/s.

Authorities Most of the analysis is based on the period between 2018-05-18 and 2018-06-01. During this time, nine DAs named "bastet", "dannenberg", "dizum", "faravahar", "gabelmoo", "longclaw", "maatuska", "moria1", and "tor26" were active. Four of those (bastet, faravahar, maatuska, moria1) were also participating as BAs. An additional *Bridge Authority* named "Serge" offers information about further RNs not listed in the public consensus which can be used if governments try to block access to Tor by restricting communication to Tor nodes listed in the consensus. Additional information and a short history of the DAs is presented in [13].

Frequency The consensus consistently lists the majority of RNs. From the 5975 different RNs included in the 1263 BA votes [18] between 2018-05-18 and 2018-06-01, 4776 RNs (79.9%) were included

[18] Note that one vote from the BA moria1 was not published.

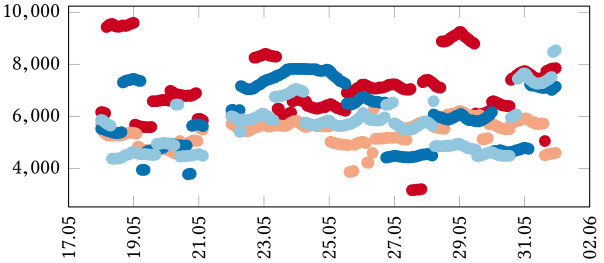

Figure 13: Bandwidth Votes for the RN "Karai" in KB/s (y-axis) by the BAs —●— bastet, —●— faravahar, —●— maatuska, and —●— moria1 over time (x-axis).

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
DA_+	1048	3160	5270	5860	6072.1	6890	9590
$DA_\bullet$	1259	3160	5110	5820	6028.2	6880	9590

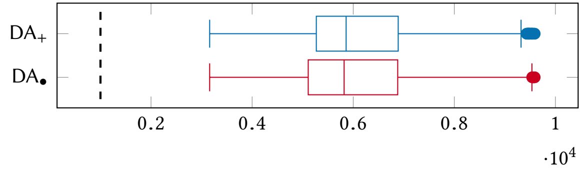

Figure 14: DA vote sets for "Karai" (1000KB/s) in KB/s.

in more than 1200 votes. Since two measurements are unlikely to result in the same value, one can assume that voting for the same measured bandwidth more than once in a row is equivalent to this relay not being measured again. Measurement results indeed are commonly reused in the votes as 4087 (68.4%) of the RNs had less than 126 changes of the measured bandwidth value during this period.

Quality Example The speedracer measurement results are far-flung even for frequently measured relays. Figure 13 shows the bandwidth votes for the relay "Karai" which received the highest amount (1048) of different bandwidth votes in the given period. As shown in Figure 14, this RN was assigned far more bandwidth than it advertised.

Quality Numbers Bandwidth votes of different BAs do not necessarily fit to each other. Given the five ordered bandwidth votes $m_1 \leq m_2 \leq ... \leq m_5$ of a relay, we define the bandwidth scattering value $bs := \frac{m_4 - m_2}{m_3}$. The average (0.50) and median (0.41) scattering value in the in the 2017-11-01 22:00:00 [19] votes show that the bandwidth votes regularly differ between 50%-150% from the assigned consensus value.

Analysis Due to the measurement over two hop circuits, the measurements are depended on similarly sized relays put in the same group. Two consecutive measurements from the same BA, most of the time, result in a similar value. Sudden changes in the order of magnitudes during the measurement most likely are caused by the relay being put into another group during the slicing of the network. This process most likely also is responsible for the fact that different BAs produced results in a different order of magnitude. Since there is a relay life cycle [14], the time the relay registered to the DAs also influences the measurement result. Because of this, measurements do not necessarily reflect the actual amount of bandwidth a Tor client can expect from a particular relay.

Conclusion Most relays are measured infrequently, and the measurement results are far-flung. Among other variables, the slicing of the network seems to be the most significant influence on this behavior.

C SMART CONTRACT IMPLEMENTATION

We show the full system model of the SC including stored information, implemented structures, and possible interaction in Figure 26. The current state of a BM, depicted in Figure 9 limits the interaction it can have with the SC and is changed by certain events. We describe the possible states in more detail below.

UNKOWN addresses do not belong to an entity yet. In order to save storage, the SC looks up the state of an address and assume it to be UNKOWN if no entry is present. UNKOWN addresses may query the state of the SC and can register themselves as RN or BM, but cannot interact with the SC in any other way.

REGISTERED BMs are the active group of BMs who may join in measuring rounds, send reports or vote for a new measurer to join. Addresses are only added to the list of REGISTERED measurer if a genuine enclave created the private key used to sign the registering transaction. The SC rejects any request to join a measurement group or vote on accepting a new BM or if the address is not REGISTERED.

KNOWN In order to get the status REGISTERED, a BM first gets the status KNOWN after it sends the registering transaction τ_{BM} to the SC. During this time, the REGISTERED BMs may vote for this entity. The SC adds the BM to the REGISTERED group if it passed verification while rejecting any votes for BMs who are not KNOWN.

UNTRUSTED If a BM fails to pass verification, it gains the status UNTRUSTED. The SC does not remove it from the blockchain in order to save its quote and prevent rejoining of its BM-TEE with another address. After a certain amount of time, this entry is deleted, giving the BM-TEE the ability to register again. This also saves storage if registered addresses are not associated with a TEE.

EXPIRED If the BM does not manage to report valid bandwidth until the group timed out, it keeps the status MEASURING. However, since the group timed out, it neither can report measurements nor join a new measurement group. In this situation, it can expire itself. If EXPIRED, a BM may request re-joining. In this case, the SC resets its status to KNOWN. If the SC decides to kick a relay out, it sets the BM to EXPIRED rather than UNKOWN in order to keep the information associated with this BM, i.e., the reputation score.

D MEASUREMENT DATA

In this section, we depict the measurement results for all RNs we measured. Figures 15 to 20 show the measurement results of the set RN_+. Figures 21 to 25 show the results of the RNs which did not produce an expected value with either script. In these sets, the RNs are ordered by advertised bandwidth as first and name as second sorting criteria.

[19] We chose this point of time as there only are four active BAs in the other evaluation period. General behavior did not change since then as indicated in the example before.

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
SMS	976	32	90	93	90.4	94	99
DA_+	103	16	25	29	29.5	34	43
$DA_\bullet$	1147	16	25	29	29.4	34	43

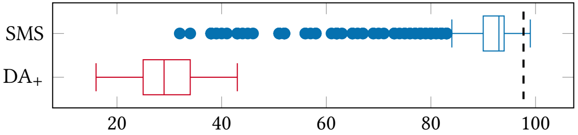

Figure 15: Results for “gnarzkorf2” (97.7) in KB/s.

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
SMS	975	12	91	94	91.1	96	99
DA_+	95	15	25	29	30.8	38	47
$DA_\bullet$	1259	15.0	25.0	27.0	29.8	34.0	47.0

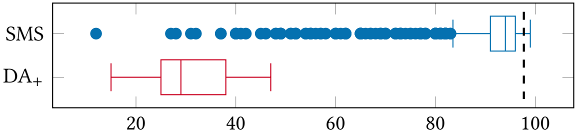

Figure 16: Results for “greedygertie” (97.7) in KB/s.

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
SMS	977	31	89	91	89.0	94	99
DA_+	88	11	22	29	28.6	36	43
$DA_\bullet$	1255	11	23	27	27.8	33	43

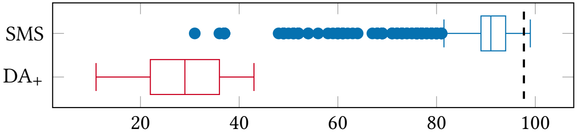

Figure 17: Results for “seele” (97.7) in KB/s.

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
SMS	588	30	193	233	208.1	241	251
DA_+	53	106	137	154	160.0	177	229
$DA_\bullet$	1254	106	141	163	165.2	201	229

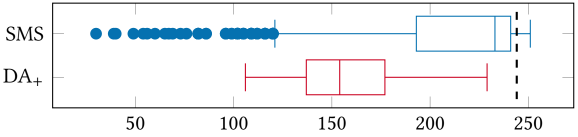

Figure 18: Results for “tiger” (244.1) in KB/s.

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
SMS	979	19	261	350	314.3	377	406
DA_+	82	81	348	428	443.5	527	775
$DA_\bullet$	1263	81	348	428	428.7	488	775

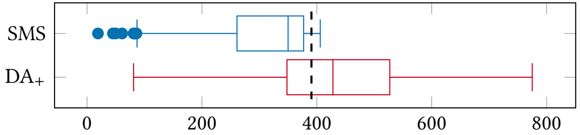

Figure 19: Results for “helga” (390.6) in KB/s.

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
SMS	977	17	188	225	204.8	233	317
DA_+	92	353	611	716	812.0	958	1690
$DA_\bullet$	1263	353	596	684	774.4	912	1690

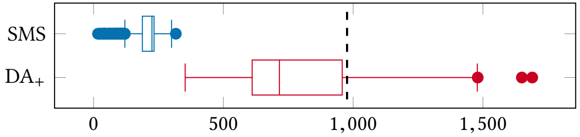

Figure 20: Results for “KAMIKAZE” (976.6) in KB/s.

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
SMS	978	23	270	785	783.7	877	2165
DA_+	87	17	175	214	226.9	293	473
$DA_\bullet$	1243	17	155	220	228.7	306	473

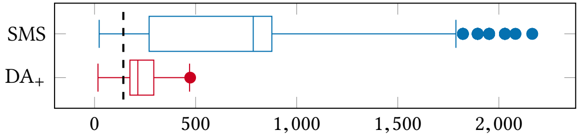

Figure 21: Results for “kwail” (142.4) in KB/s.

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
SMS	978	53	225	362	415.3	604	911
DA_+	75	576	1250	1420	1389.6	1590	1900
$DA_\bullet$ 1255	576	1220	1430	1392.3	1600	1900	

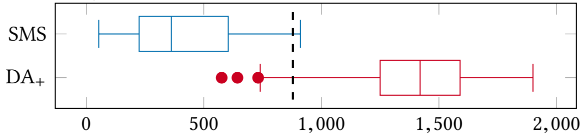

Figure 22: Results for “Ograoum” (878.9) in KB/s.

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
SMS	977	31	376	541	550.9	719	1006
DA$_+$	86	176	1330	1680	1640.3	1960	2710
DA$_\bullet$	1263	176	1340	1700	1639.3	1940	2710

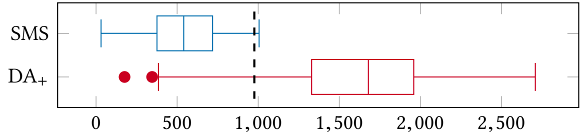

Figure 23: Results for "kolben" (976.6) in KB/s.

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
SMS	973	18	241	362	332.2	436	605
DA$_+$	88	334	1140	1270	1423.5	1440	8210
DA$_\bullet$	1263	334	1100	1230	1424.1	1380	8210

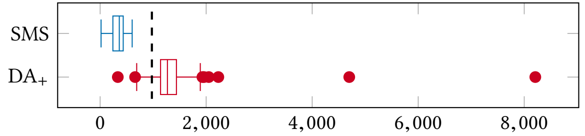

Figure 24: Results for "lkjhgfdsa" (976.6) in KB/s.

set	n	min	$n_{0.25}$	med	avg	$n_{0.75}$	max
SMS	978	29	406	541	554.6	719	981
DA$_+$	68	621	1320	1510	1505.8	1760	2400
DA$_\bullet$	1263	621	1210	1560	1500.3	1750	2400

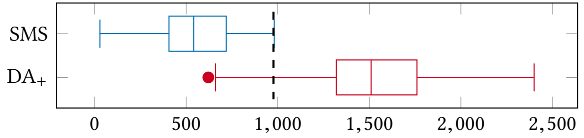

Figure 25: Results for "panic" (976.6) in KB/s.

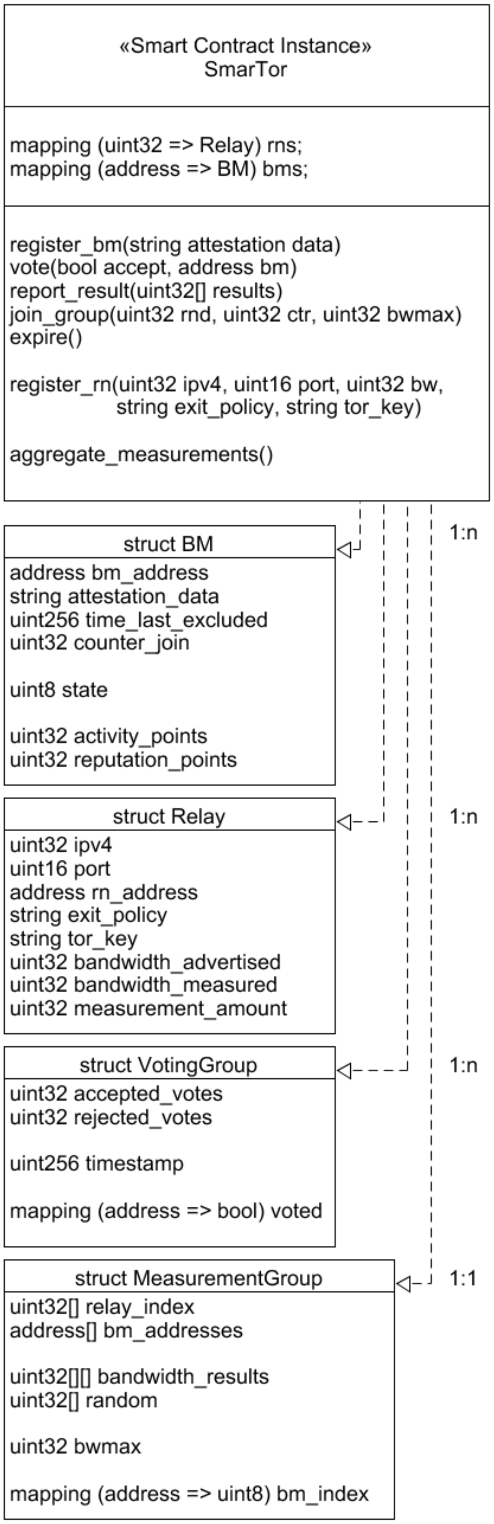
«Smart Contract Instance»
SmarTor
mapping (uint32 => Relay) rns;
mapping (address => BM) bms;
register_bm(string attestation data)
vote(bool accept, address bm)
report_result(uint32[] results)
join_group(uint32 rnd, uint32 ctr, uint32 bwmax)
expire()
register_rn(uint32 ipv4, uint16 port, uint32 bw,
string exit_policy, string tor_key)
aggregate_measurements()
struct BM
address bm_address
string attestation_data
uint256 time_last_excluded
uint32 counter_join
uint8 state
uint32 activity_points
uint32 reputation_points
1:n
struct Relay
uint32 ipv4
uint16 port
address rn_address
string exit_policy
string tor_key
uint32 bandwidth_advertised
uint32 bandwidth_measured
uint32 measurement_amount
1:n
struct VotingGroup
uint32 accepted_votes
uint32 rejected_votes
uint256 timestamp
mapping (address => bool) voted
1:n
struct MeasurementGroup
uint32[] relay_index
address[] bm_addresses
uint32[][] bandwidth_results
uint32[] random
uint32 bwmax
mapping (address => uint8) bm_index
1:1

Figure 26: System overview of the SC implementation.