Master of Puppets: Trusting Silicon in the Fight for Practical Security in Fully Decentralised Peer-to-Peer Networks

Bernd Prünster[1,2][a], Edona Fasllija[1,2] and Dominik Mocher[1,2]
[1]*Institute of Applied Information Processing and Communications (IAIK), Graz University of Technology, Austria*
[2]*Secure Information Technology Center – Austria (A-SIT), Austria*

Keywords: Peer-to-Peer Security, Decentralised System Security, Hardware-based Security, Trusted Computing.

Abstract: This paper presents a practical solution to Sybil and eclipse attacks in a fully decentralised peer-to-peer context by utilising trusted computing features of modern Android devices. We achieve this by employing hardware-based attestation mechanisms introduced in recent Android versions and bind each P2P network node identifier to a distinct physical device. In contrast to resource-testing approaches, this binding makes it impossible for attackers to rely on cheap cloud computing resources to outperform legitimate users. We address well-known P2P challenges by applying trusted computing approaches, which were previously only theorised in this context. This results in a system that can now actually be implemented on a global scale. We thoroughly mind bandwidth, power and performance constraints to achieve a ready-to-use solution whose only requirement is the possession of a recent Android phone.

1 INTRODUCTION

Practical security in fully decentralised *peer-to-peer* (P2P) networks has remained a challenge ever since such systems have been proposed. As centralised peer-to-peer security is a non-issue, this paper focuses on decentralised P2P systems. *Sybil* Douceur (2002) and *eclipse* attacks Singh et al. (2004) are still prominent threats to peer-to-peer systems and require elaborate countermeasures. The advent of cheap cloud computing power even worsened this situation, rendering resource testing approaches virtually ineffective (Prünster et al., 2018). In fact, Prünster et al. argue that the only effective countermeasure is to either introduce some degree of centralisation or couple the P2P network layer tightly to the application running on top of it.

We present *the Master of Puppets* as a practical solution to the most critical security issues in fully decentralised scenarios. The Master of Puppets' approach does, for the first time, not involve inefficient resource testing or centralisation. Our solution follows a mobile-first approach targeting current Android devices and builds upon hardware-based key attestation capabilities and self-certifying identifiers to effectively eliminate eclipse and Sybil attacks under real-world conditions. In essence, recent advancements of Android's remote attestation capabilities have made it possible to actually utilise trusted-hardware-based security measures while still providing a practically applicable solution.

In short, we present the first practical P2P network design which can actually enforce every device to operate only a single peer. This rules out both Sybil and eclipse attacks and thus eliminates the foundation of whole classes of peer-to-peer-specific attacks. We accomplish this without requiring elaborate enrolment procedures, centralised identifier generation, inefficient resource testing like *proof-of-work* (PoW).

The following section covers the technological background of our concept and summarises the state of P2P security and Android's security features to illustrate how they can be used to enforce operating unmodified applications on actual physical devices. We combine these features with self certifying identifiers and harness it as the cornerstone of the Master of Puppets. Section 3 discusses our approach, while Section 4 analyses its security aspects. Section 5 concludes this work.

2 BACKGROUND

Peer-to-peer overlays differ drastically in terms of their overlay structure, routing mechanism, how they assign identifiers, exchange data and share responsibilities. When considering security features of Peer-to-Peer networks, the specifics of the targeted P2P overlay model are

[a] https://orcid.org/0000-0001-7902-0087

Prünster, B., Fasllija, E. and Mocher, D.
Master of Puppets: Trusting Silicon in the Fight for Practical Security in Fully Decentralised Peer-to-Peer Networks.
DOI: 10.5220/0007926702520259
In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE 2019)*, pages 252-259
ISBN: 978-989-758-378-0

critical. In this paper, we focus on structured P2P overlays, and more specifically in *Kademlia* (Maymounkov and Mazières, 2002) networks, due to the many desired features and their practical impact, with *BitTorrent* (Cohen, 2013) and *IPFS* (Benet, 2014) both being based on this design.

The distributed nature of P2P systems introduces susceptibility to a series of attacks that aim at disrupting their storage, service quality, consistency, behaviour, or overall operability of the network. Examples of the most prominent attacks include routing table poisoning (Locher et al., 2010), flooding (Zargar, Joshi and Tipper, 2013), eclipse (Singh et al., 2004), and Sybil (Douceur, 2002) attacks. Certain threats, such as Sybil and eclipse attacks, are based on malicious control of identifiers. The absence of a central authority makes authentication in decentralised peer-to-peer overlays inherently difficult. Secure, verifiable, and limited node identities are essential for preventing attackers from forging node identities and staging subsequent attacks.

The Sybil attack defined by Douceur (2002) is based on the concept of inserting a node into a P2P network multiple times, each time with a different identity. The goal of Sybil attack might range from enabling other attacks to disrupting the network connectivity, or even effect majority decisions in consensus protocols. Levine, Shields and Margolin (2006) categorise approximately 90 approaches that have been proposed as defense mechanisms against Sybil attacks. These fall into categories such as: trusted certification, resource testing, no solution, recurring costs and fees and trusted devices. A running theme among these approaches is the idea that without a central trusted authority that certifies peers' identities, there is no realistic approach to prevent Sybil attacks. Resource-testing based approaches mandate that all peers—even highly-constrained ones—have to spend a certain amount of resources, such as a *proof-of-work* (PoW) to have the network accept them. Given the heterogeneous distribution of computing power, storage and bandwidth in today's diverse device landscape and the cheap availability fo cloud computing, the utility of such approaches is questionable (Prünster et al., 2018).

Eclipse attacks (Singh et al., 2004), on the other hand, target a node or a specific group of nodes and try to isolate (or 'eclipse') them by populating their first-hop neighbor set. The goal is to control a victim's neighbours as wells as incoming and outgoing traffic. Attackers might first stage a Sybil attack in order to create seemingly distinct malicious nodes around attack targets, or directly mount an eclipse attack using a small set of malicious nodes.

Various countermeasures have been proposed to enhance the resiliency of P2P overlays against eclipse attacks. Baumgart and Mies (2007) propose a mitigation mechanism against, amongst others, eclipse attacks, on Kademlia networks. They base their proposal on self-certifying identifiers combined with proof-of-work-based identifier generation and disjoint key lookup paths. Their cryptographic scheme for signatures used to authenticate their nodes relies either on a crypto puzzle, or a central certificate authority. The latter is required since their security model is based on effective defences against Sybil attacks. Other similar strategies against eclipse attacks base their mitigation schemes on a centralized encryption authority. For example, Castro et al. (2002) solve the problem of secure node ID assignment by making use of a set of central trusted authorities (CAs) to assign and sign node IDs when joining the network. Likewise, Fantacci et al. (2009) enhance the Kademlia architecture and protocol by employing external certification services to bind node IDs to a public key in a token.

Some of the proposed mitigation approaches pose additional structural or proximity constraints on neighbour selection. Nonetheless, they still rely on a trusted authority for the issuance of unique node IDs. Hildrum and Kubiatowicz (2003) propose to defend P2P systems against eclipse attacks by enforcing neighbour node selection based on the minimum network delay. However, these approaches negatively affect the performance by introducing additional overheads and impeding optimisations.

Having provided an overview about peer-to-peer security measures, we now discuss Android security, as the second pillar of our approach.

2.1 Android as Trusted Computing Base

Since Android security in general has been discussed extensively in existing literature and is well documented by Google, we only recap the basics and focus on recent features, such as key attestation, that elevate Android smartphones to a *trusted computing base* (TCB).

2.1.1 Android Security

To prevent applications from accessing arbitrary data, the Android operating system uses sandboxing in combination with mandatory access control. The former concept is realised by assigning a unique user ID to each process, while the latter relies on *SELinux* (Android Source, 2019b). As sandboxing is implemented purely in software, a compromised or altered OS (like a custom ROM) allows for circumventing these restrictions. With the introduction of *Trusty* (Android Source, 2019c) Google has taken action to prevent access to cryptographic keys in such scenarios by relying on trusted hardware.

Since Trusty is a *trusted execution environment* (TEE) designed to perform only predefined tasks, it is isolated from the system by hardware and software, meaning interaction from the operating system is only possible through a dedicated interface. With the absence of any means to export keys, cryptographic binding to hardware is enforced at the cost of limiting the TEE to operations defined by the interface. With the support of *hardware security modules* (HSMs) in Android 9, even stronger security guarantees are provided. Smartphones equipped with such an HSM must have it "certified against the Secure IC Protection Profile BSI-CC-PP-0084-2014 or evaluated by a nationally accredited testing laboratory" (Android Source, 2019a, p. 128). This essentially certifies the HSM as tamper-proof hardware that protects key material to a degree equivalent to smart card standards.

Although keys might not be extractable, managing key access is managed by the operating system and thus relies on the integrity of the system. System integrity is ensured by a verified boot process, which is heavily dependent on trusted hardware and thus cannot be circumvented using software-only mechanisms.

2.1.2 Remote Attestation

Even though the verified boot process ensures system integrity, the user might still be able to influence the execution of an application at runtime, as static or dynamic code integrity checks are not an integral part of Android. Software-only attestation mechanisms like SafetyNet (Android Open Source Project, 2018) can be circumvented by rooting tools like *Magisk*[1], thus demonstrating the need for a reliable attestation mechanism implemented in hardware. As a requirement for this kind of attestation, the device has to be outfitted with a TEE or HSM. As mandated by Google, these hardware modules are preloaded with an X.509 certificate and a signing key during the manufacturing process (Android Source, 2019a). We refer to this certificate as the *attestation certificate* and the *attestation key*, respectively. In addition, a chain leading up from the signing certificate to a Google-signed root certificate (which is publicly available) is provisioned during the manufacturing process.
Harrware-based remote attestation thus makes it possible to verify that cryptographic keys created by an application are actually stored in hardware. Moreover, recent Android versions (8.0 and later) enable attesting app integrity, which is a key enabler of trusted computing. This process works as follows: The HSM/TEE queries verified boot and bootloader lock state, as well the hash of the certificate that was used to sign an application. The output of this attestation process is a certificate, signed by the attestation key in hardware and then returned to the application, containing all queried information as certificate extensions. Thus, it becomes possible to remotely verify these values. If the trust chain up to the Google-signed root certificate holds, the bootloader state is attested as *locked* and the system image as *verified*, we can assume an uncompromised device. As for verifying app integrity, the mentioned hash of the app's signature certificate can simply be checked against a precomputed hash value (typically published by the app's developer). Thus, attested applications run on a TCB and generated cryptographic material is securely stored in hardware and therefore bound to a physical device.

Almost all upcoming devices with Android 8.0 or later are expected to support hardware-backed key attestation according to the FIDO Alliance (FIDO Alliance, 2018). This kind of hardware-based attestation enables the practical implementation of security models previously proposed only in theoretical scenarios. The following section introduces the Master of Puppets and explains how it utilises this fact to deliver practically secure P2P networking in a fully decentralised context.

3 THE MASTER OF PUPPETS

This section presents our approach towards practically secure, fully decentralised P2P networking by binding the creation and operation of peer-to-peer nodes to trusted devices as discussed in Section 2.1.2. We present how this previously purely theoretical thought experiment can be transformed into a practically feasible solution. We achieve this by harnessing already deployed Android devices whose trusted state can be verified remotely.

Our solution is based on the *S/Kademlia* (Baumgart and Mies, 2007) extension of the *Kademlia* structured P2P network and supports the following operations:

Ping: Probes a node to see whether it is online

Lookup: Queries the network for IP address and port of a node based on its identifier.

Store: Stores some data among the set of nodes whose identifiers are closest to the data's hash

Find: Retrieves some data based on its hash

We rely on Android's remote attestation capabilities to defend against Sybil attacks. Therefore, a recent Android device is needed to operate a network node. Thus, we require an Android application to be run on a suitable device as defined in the following section.

3.1 Targeted Devices

We base our definition of the targeted device set on the features discussed in Section 2.1. Devices must feature a hardware-backed keystoreand must provide all remote attestation capabilities introduced by Android 8.0.

[1]https://topjohnwu.github.io/Magisk/

While we have validated that these requirements rule out some devices that were released with older Android versions that received an 8.0+ update, we still conservatively estimate at least 100M suitable devices deployed as of February 2019. This figure is based on matching Android version market shares against sales numbers of popular handsets: Samsung's *Galaxy S9* device range alone accounts for $>$ 45M devices (Statista, 2019). Huawei's *P20* family has sold an additional 16M[2], and its *Mate 20* series $>$ 10M[3]. Moreover, if we estimate a typical device life time at two years, compatibility issues will become irrelevant in the foreseeable future.

3.2 System Model

As the Master of Puppets is based on Kademlia, it shares most of its basic characteristics, like the XOR distance function and the iterative node lookup procedure. We therefore do not reiterate about these basic properties at this point and only describe where our system deviates from the vanilla Kademlia design.
First and foremost, we borrow the core ideas from S/Kademlia and mandate self-certifying identifiers based on public key cryptography as well as disjoint paths for node lookups. We do not modify the original Kademlia lookup procedure in any other way.

In essence, each message contains the identifier of the sender, which is bound to the possession of a private key that is used to sign this message. Thus, anyone can verify the origin of each message. As initially stated, our defence against Sybil attacks is based on Android's remote attestation features.

Apart from identifiers binding to an Android device, this design aligns with the security concepts outlined in Section 2 and results in a robust P2P network, as long as a defence against Sybil attacks is in place.

3.3 Remote Attestation-based Identifier Preliminaries

As our design mandates Android-device-bound keys for identifier creation, we discuss the process of obtaining a self-certifying identifier and its structure. We then explain how this can be integrated into the above system model and argue why this is practically feasible.

The foundation of identifier creation is a suitable Android device as described in Section 3.1 (equipped with either a TEE or and HSM). Our system therefore requires an Android app to generate a public/private key pair using Android's hardware-backed keystore implementation and have it attested. This results in the following data:

- App metadata (including app signature certificate's hash)
- Key metadata (algorithm, public key fingerprint, ...)
- Bootloader lock state (one of *locked*, *unlocked*)
- Verified Boot state (one of *Verified*, *SelfSigned*, *Unverified*)

These values are encoded into an X.509 certificate as extensions and signed using the attestation key provisioned into the trusted hardware during the manufacturing process. The signed certificate, including the certificate chain leading up to the root certificate published by Google, is returned to the application.

If we mandate the Google root certificate as root of trust on all nodes, we can validate the authenticity of such an attestation result. As we aim for a fully decentralised design, this happens offline, and no third party is involved in this process. By checking whether the boot chain is *Verified* for a system running on a *locked* device, we can make sure that the Master of Puppets' Android app is running on a trusted device. By comparing the hash of its signature certificate contained in the attestation result, we can further ensure that an unmodified version of the app is used, thus establishing trust in installed instances remotely.

By deriving a node's identifier from its attestation result, we can ensure that the identifier of each node in the Master of Puppets's P2P network is bound to a physical, trusted device and thus prevent the creation of multiple identifiers per device. The following section provides detailed information on how this is achieved.

3.4 Preventing Sybil Attacks through Device-bound Identifiers

Applying the proposed process to the system model defined in Section 3.2 aligns with the concept of self-certifying identifiers, since receivers can directly validate the authenticity of every incoming message by matching its signature against the included identifier. Yet, this still leaves one attack vector: Android enables the creation of multiple user accounts per device—each with its own set of apps and data. To prevent this scenario, we designed our Android app such that it also includes the hash of a device's *International Mobile Equipment Identity* (IMEI) (or *mobile equipment identifier* (MEID), in case of a CDMA device) as part of the self-certifying identifier. Thus identifiers in our system are computed as `SHA3-256(SHA3-256($IMEI/$MEID)||$attestation_cert)`.

[2]https://consumer.huawei.com/en/press/news/2018/huawei-annual-smartphone-shipments-exceed-200-million-units/

[3]https://consumer.huawei.com/en/press/news/2019/huawei-mate-20-series-shipments-exceed-10-million-units/

A comprehensive discussion about the security properties of our approach is provided in Section 4 that deals with our system as a whole, and also discusses attack costs compared to existing solutions relying on resource testing. Before going into more detail about security properties, we present the remaining details of our system.

3.5 Efficiency and Overhead

Relying on a whole certificate chain that can easily exceed `1kB` for messages whose payload is often less than `100B` is not practical. We utilise a key property of the certificate chain to remedy this issue, originally aimed at preventing device identification: Google mandates that no less than 100k devices share an intermediate certificate (Android Source, 2019a). Therefore, our design only mandates the attestation certificate to be present in messages as the self-certifying identifier and omits the full chain. Instead, receivers can enquire the full chain if they receive a message whose signature cannot be verified due to missing intermediate certificates. Each node is obliged to store all certificate chains received, which, in time, leads to all nodes learning all chains.

Matching this approach even against all > 2.3 billion active Android devices (van der Wielen, 2018), $\sim 21\%$ of which are running Android 8+[4] (the majority of which are not within the set of targeted devices) still only results in ~ 5000 possible certificate chains. This approach reduces the overhead introduced through the Master of Puppets to less than `256B` per message.

3.6 Architecture

Although our design mandates the use of an Android app, this does not necessitate burdening resource-constrained mobile devices with the operation of network nodes. Instead, the Master of Puppets offers two distinct modes of operation:

Mobile-only: Using only the Android app to operate a node directly on a mobile device.

Phone-bound: Outsourcing only identifier generation and message signing to the Android app from another device (typically a PC or a laptop) operating the actual node. For the remainder of this paper, we refer to this device as the *host*.

The *mobile-only* mode of operation is self-explanatory from an architectural point of view. Therefore, we do not focus on it and instead elaborate on the phone-bound mode of operation and its implications.

[4]https://developer.android.com/about/dashboards/

3.6.1 Universal Trust using Phone-bound Nodes

By only relying on a mobile device running the Master of Puppets' Android app for cryptographic operations involving private key material, the device becomes not much more than a universally trusted smartcard. Compared to previously theorised approaches using smartcards or trusted computing concepts, we solve the trust problem by relying on devices that are already trusted on a global scale. The Android app is designed such that it only supports a single host, to still uphold the premise of requiring a distinct device for each node. To achieve this, we mandate an authenticated channel between host and phone.

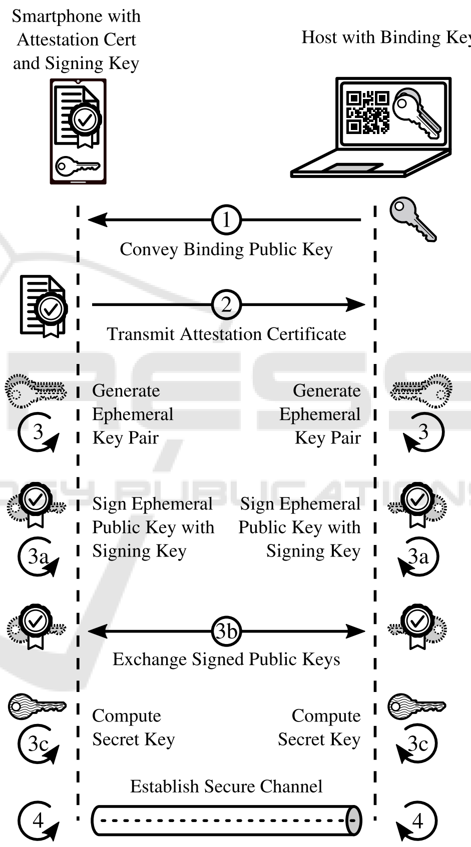
Smartphone with Attestation Cert and Signing Key
Host with Binding Key
1
Convey Binding Public Key
2
Transmit Attestation Certificate
Generate Ephemeral Key Pair
3
Generate Ephemeral Key Pair
3
Sign Ephemeral Public Key with Signing Key
3a
Sign Ephemeral Public Key with Signing Key
3a
3b
Exchange Signed Public Keys
Compute Secret Key
3c
Compute Secret Key
3c
Establish Secure Channel
4
4

Figure 1: Binding host to phone and establishing a mutually authenticated channel using ECDHE key agreement.

3.6.2 Authenticated Phone-host Channel

We accomplish mutual authentication based on an *Elliptic Curve Diffie-Hellman ephemeral* (ECDHE) scheme to efficiently establish authenticated, end-to-end encrypted communication between the Android app and the host (see Figure 1). The phone is identified using the private key matching its attestation certificate. We refer to this key as the *signing key*. The host, on the other hand, generates a public/private key pair. This is called the *binding key*. Binding the host to the phone works as follows:

1. The host's public binding key is convened to the phone, by scanning a QR-code off the host's screen.
2. The attestation certificate is transmitted to the host.
3. Having established a binding, both phone and host generate an ephemeral public/private key pair to engage in an ECDHE key agreement process to create a mutually-authenticated communication channel:
   (a) The phone signs its ephemeral public key using its signing key and the host signs its ephemeral public key using its private binding key.
   (b) The signed ephemeral public keys are exchanged.
   (c) Both parties compute a secret key based on their ephemeral private keys and the received signed ephemeral public keys (using ECDH).
4. Phone and host establish a communication channel based on the previously computed secret key.

For the actual transmission of message digests and signatures values, we employ the *ChaCha20* (Nir and Langley, 2018) stream cipher and use the secret key agreed upon between host and phone during the ECDHE key exchange. Utilising a stream cipher does, by definition, not introduce any payload overhead, once a communication channel has been established. Consequently, throughput even on low-bandwidth links (like Bluetooth *Enhanced Data Rate* (EDR)) is not impaired. We mandate re-keying every 2^{32} messages as a middle ground between security margin and overhead.

Most probably, the key material created on the host will not be stored in hardware and is thus extractable and can be replicated. This, however, has no impact on the overall security of our network design: Binding the host to a phone is only required to prevent unauthorised third parties from utilising the phones of others. Even if a malicious entity copies their binding key to a huge amount of host devices, signing messages would still be done by a single phone. Consequently, this entity would still only present a single identity to the network.

Having means in place to establish an authenticated, encrypted communication channel between the host and the phone, the characteristics of the transport layer regarding these properties become irrelevant.

3.6.3 Efficiently Outsourced Signing

From a technical point of view, we mandate a Bluetooth EDR (Bluetooth SIG, 2019) or WIFI connection between the Android app and the host. Although Bluetooth EDR is a low-bandwidth link, it is still perfectly adequate if we reduce the traffic required to operate a network node to a minimum. We achieve this as follows:

Firstly, we implement detached signing and detached signatures as depicted in Figure 2 (assuming an authenticated channel between host and phone is in place): The phone (1) hashes its IMEI, and the host (1) hashes the message to be sent, and (2) only transmits the hash to the phone. The phone then creates a detached signature by (3) combining the received hash with its hashed IMEI, and by (4) directly applying the signature operation to this data (thus skipping the usually required hashing operation during signature creation). The signed data is then (5) returned to the host where it is (6) appended to the message, resulting in a signed message that is bound to an individual phone.

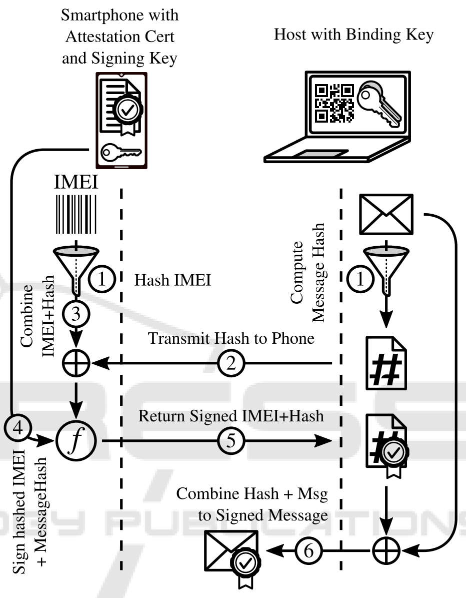

Figure 2: Creation of a signed message.

Secondly, validation of incoming messages happens on the host and thus puts no strain even on a Bluetooth EDR connection. This way of producing signed messages addresses the basic process carried out for the Kademlia operations (*Ping*, *Lookup*, *Store*, and *Find*).

4 SECURITY CONSIDERATIONS

Due to its conceptual resemblance of S/Kademlia, our system inherits its overall security properties. Consequently, like S/Kademlia, our design remains secure, unless identifiers can be chosen freely and Sybil attacks can be carried out. In general, we do not tackle implementations flaws and deficiencies of cryptographic primitives and consider such flaws out of scope. By extension, we also do not cover the area of root exploits on Android. While this may seem like a limitation, it simply entails that we trust the platform the Master of Puppets is developed for.

4.1 On Phone-bound Identifiers

As presented in Section 3.4, the Master of Puppets' identifiers are derived by combining an attestation result, created by trusted hardware with the hash of a device's IMEI or MEID and then have these values signed by the key referenced in the attestation result. Since the attestation result contains a proof that an unmodified app has been used (in the form of the hash of the certificate that was used to sign the app), identifiers cannot be spoofed. Consequently, any other node in the network can remotely verify that an unmodified application was used. Since this happens on every incoming message, it becomes impossible to establish the presence of compromised nodes in the network. Although the software running on the host is not considered trusted, this has no impact on our security model, since it aligns perfectly with S/Kademlia. Thus, the security analyses performed on this design can also be applied to the Master of Puppets.

4.2 Attack Costs

By binding an identifier to the possession of a distinct device, the cost for identifier generation and node operation can be estimated in a straight-forward manner. The cheapest phone we know of (that supports all required attestation features) is the *Nokia 1*, currently retailing at €79.00 on Amazon.de. We compare this price tag for node operation to the figures obtained by Prünster et al. (2018), who analysed the cost of Sybil and eclipse attacks for proof-of-work-based identifier creation in a hardened S/Kademlia network, based on the price of cloud computing resources. As these prices have not significantly changed since, these 2018findings still hold.

4.2.1 Sybil Attack Costs

Prünster et al. reached reached the conclusion that creating a million identifiers, even in a hardened S/Kademlia design mandating a PoW could be priced at roughly some few thousand Euro, while operating this amount of nodes would incur running costs of about €33/h for a network of 1M nodes. If we map this scenario to our approach and optimise attack costs by relying on cheap Nokia 1 phones, operating a million nodes, we arrive at initial costs of €79 000 000. Running costs are essentially depended on the cost of electricity.

4.2.2 Eclipse Attack Costs

Successfully eclipsing a target node in a network of a million nodes requires generating some tens of millions of nodes, as identifier generation using the SHA3 cryptographic hash function is unpredictable due its output being pseudorandom. In short, a brute-force approach is the best possible way to obtain the desired identifiers. We again refer to Prünster et al. (2018) for concrete figures.

Based on this, we can estimate the time it would take to generate and test 20M identifiers to eclipse a single target in a network of 1M nodes. We assume an attacker possesses 20 devices (to be able to operate 20 nodes). Each node generation process requires discarding the previously generated identifier by clearing the data of the Master of Puppets' App and re-initialising it. If we optimistically assume a duration of 100ms for this process and have it run in parallel on all 20 devices, it would take roughly one day to eclipse a target (see Eq. 1).

$$t_{ecl} = \underbrace{20 \times 10^6}_{\text{\#IDs to try}} \times \underbrace{0.1/3600}_{\text{time per ID in hours}} \; / \; \underbrace{20}_{\text{\# of phones}} \sim 28\text{h} \quad (1)$$

This is perfectly feasible and does not incur high initial or running costs. However, it is impossible to outsource this process, as identifiers are bound to cryptographic keys located inside the phone's trusted hardware. Obviously, our solution would fail to defend against eclipse attacks, without other countermeasures in place. We therefore introduce a delay of 1 minute into the identifier generation process. Since this delay is executed on the phone as part of the remotely attested Android application, it also cannot be sped up. As this results in a slow-down factor of 600, generating and testing enough identifiers to eclipse a single node in a network of 1M nodes would then take approximately two years.

Even if a malicious entity owns hundreds of devices to mount an eclipse attack, the victim could simply generate a new identifier and thus be out of the attacker's reach. It is important to note that mounting another eclipse attack on the new identifier would take the same mount of time than the initial attack.

5 CONCLUSIONS

This work presented the Master of Puppets as a practically secure approach towards fully decentralised P2P networking. Our design utilises current Android smartphones as universally trusted smartcards and binds the operation of P2P network nodes to the possession of a physical device. Compared to resource testing approaches like PoW, this prevents attackers from utilising cheap cloud computing offers to outperform legitimate users. We base our system on the S/Kademlia P2P network, adapt its self-certifying identifiers to utilise Android's key attestation framework to bind identifiers to the possession of a physical device. In a nutshell, this means that a distinct device is required to operate a peer-to-peer network node. This effectively combats Sybil and eclipse attacks, and shows how it is now, for the first time, possible to apply trusted computing to provide secure P2P networking without relying on a central authority.

Although this limits the potential user base to owners of supported devices, the popularity of Android still makes our approach usable by millions of users. As our design outsources only some cryptographic operations to phones, even low-end devices can be used. Most importantly, however, we have shown that it is now, for the first time possible to utilise a widely available trusted computing base to effectively combat eclipse or Sybil attacks without relying on inefficient resource-testing approaches that can easily be circumvented due to the low prices of widely-available cloud computing resources.

REFERENCES

Android Open Source Project (Apr. 2018). *Protecting against Security Threats with SafetyNet*. U R L: https://developer.android.com/training/safetynet/ (visited on 11/01/2019).

Android Source (Feb. 2019a). *Android 9.0 Compatibility Definition*. U R L: https://source.android.com/compatibility/9/android-9-cdd.pdf (visited on 20/02/2019).

— (2019b). *Security-Enhanced Linux in Android*. U R L: https://source.android.com/security/selinux/ (visited on 19/02/2019).

— (2019c). *Trusty TEE*. U R L: https://source.android.com/security/trusty/ (visited on 11/01/2019).

Baumgart, I. and S. Mies (Dec. 2007). 'S/Kademlia: A Practicable Approach towards Secure Key-Based Routing'. In: *2007 International Conference on Parallel and Distributed Systems*, pp. 1–8. D O I: 10.1109/ICPADS.2007.4447808.

Benet, Juan (July 2014). *IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)*. U R L: https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf (visited on 04/01/2019).

Bluetooth SIG (Jan. 2019). *Bluetooth Core Specification v5.1*. U R L: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=457080 (visited on 09/03/2019).

Castro, Miguel et al. (2002). 'Secure Routing for Structured Peer-to-peer Overlay Networks'. In: OSDI '02, pp. 299–314. U R L: http://dl.acm.org/citation.cfm?id=1060289.1060317.

Cohen, Bram (Oct. 2013). *The BitTorrent Protocol Specification*. U R L: http://www.bittorrent.org/beps/bep0003.html visited on 24/04/2017).

Douceur, John R. (2002). 'The Sybil Attack'. In: *Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek and Antony Rowstron. Vol. 2429. Lecture Notes in Computer Science. Berlin, Heidelberg, Germany: Springer, pp. 251–260. I S B N: 978-3-540-44179-3.

Fantacci, R. et al. (June 2009). 'Avoiding Eclipse Attacks on Kad/Kademlia: An Identity Based Approach'. In: *2009 IEEE International Conference on Communications*. pp. 1–5. D O I: 10.1109/ICC.2009.5198772.

FIDO Alliance (June 2018). *Hardware-backed Keystore Authenticators (HKA) on Android 8.0 or Later Mobile Devices*. U R L: https://fidoalliance.org/wp-content/uploads/Hardware-backed_Keystore_White_Paper_June2018.pdf (visited on 14/01/2019).

Hildrum, Kirsten and John Kubiatowicz (2003). 'Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks'. In: *International Symposium on Distributed Computing*. Springer, pp. 321–336.

Levine, Brian Neil, Clay Shields and N. Boris Margolin (Oct. 2006). *A Survey of Solutions to the Sybil Attack*. 2006-052. Amherst, MA: University of Massachusetts Amherst.

Locher, Thomas et al. (2010). 'Poisoning the Kad network'. In: *International Conference on Distributed Computing and Networking*. Springer, pp. 195–206.

Maymounkov, Petar and David Mazières (2002). 'Kademlia: A Peer-to-Peer Information System Based on the XOR Metric'. In: *International Workshop on Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek and Antony Rowstron. Red. by Gerhard Goos, Juris Hartmanis and Jan van Leeuwen. Vol. 2429. Berlin, Heidelberg: Springer, pp. 53–65. I S B N: 978-3-540-44179-3 978-3-540-45748-0. D O I: 10.1007/3-540-45748-8_5.

Nir, Y. and A. Langley (July 2018). *ChaCha20 and Poly1305 for IETF Protocols*. RFC 8439. Published: Internet Engineering Task Force Request for Comments. U R L: https://www.ietf.org/rfc/rfc8439.txt(visitedon21/03/2019).

Prünster, Bernd et al. (2018). 'A Holistic Approach Towards Peer-to-Peer Security and Why Proof of Work Won't Do'. In: *14th EAI International Conference on Security and Privacy in Communication Networks*. Vol. 255. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Singapore: Springer, pp. 122–138.

Singh, Atul et al. (2004). 'Defending Against Eclipse Attacks on Overlay Networks'. In: *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*. EW 11. New York, NY, USA: ACM. D O I: 10.1145/1133572.1133613. U R L: http://doi.acm.org/10.1145/1133572.1133613 (visited on 21/06/2018).

Statista (2019). *Shipments of Samsung Galaxy S series smartphones worldwide from 2016 to 2018 (in million units)*. U R L: https://www.statista.com/statistics/864691/samsunggalaxy-s-series-smartphone-shipments-worldwide/ (visited on 27/02/2019).

van der Wielen, Bernd (Jan. 2018). *Insights into the 2.3 Billion Android Smartphones in Use Around the World*. Newzoo. U R L: https://newzoo.com/insights/articles/insights-intothe-2-3-billion-android-smartphones-in-use-aroundthe-world/ (visited on 28/02/2019).

Zargar, Saman Taghavi, James Joshi and David Tipper (2013). 'A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks'. In: *IEEE communications surveys & tutorials* 15.4, pp. 2046–2069.