

Sea of Lights: Practical Device-to-Device Security Bootstrapping in the Dark

Flor Álvarez, Max Kolhagen, and Matthias Hollick

Secure Mobile Networking Lab, TU Darmstadt, Germany, {falvarez, mkolhagen, mhollick}@seemoo.tu-darmstadt.de

Abstract—Practical solutions to bootstrap security in today’s information and communication systems critically depend on centralized services for authentication as well as key and trust management. This is particularly true for mobile users. Identity providers such as Google or Facebook have active user bases of two billion each, and the subscriber number of mobile operators exceeds five billion unique users as of early 2018. If these centralized services go completely ‘dark’ due to natural or man made disasters, large scale blackouts, or country-wide censorship, the users are left without practical solutions to bootstrap security on their mobile devices. Existing distributed solutions, for instance, the so-called web-of-trust are not sufficiently lightweight. Furthermore, they support neither cross-application on mobile devices nor strong protection of key material using hardware security modules. We propose Sea of Lights (SoL), a practical lightweight scheme for bootstrapping device-to-device security wirelessly, thus, enabling secure distributed self-organized networks. It is tailored to operate ‘in the dark’ and provides strong protection of key material as well as an intuitive means to build a lightweight web-of-trust. SoL is particularly well suited for local or urban operation in scenarios such as the coordination of emergency response, where it helps containing/limiting the spreading of misinformation. As a proof of concept, we implement SoL in the Android platform and hence test its feasibility on real mobile devices. We further evaluate its key performance aspects using simulation.

I. INTRODUCTION

Users of mobile applications heavily rely on third party providers to offer basic security services. In fact, practical solutions to bootstrap security in today’s information and communication systems critically depend on such third party security services for authentication as well as key and trust management. These systems are commonly designed in a centralized fashion and scale to billions of users—making them attractive for application developers/providers as well as end users, which do not want to burden themselves with alternate means of securing their apps.

If these centralized services go completely ‘dark’ due to natural or man made disasters [1], large scale blackouts [2], or country-wide censorship [3], the users are left without practical solutions to bootstrap security on their mobile devices. As a result mobile app usage is severely restricted to support users in such scenarios.

Distributed solutions to bootstrap security such as the web-of-trust (WoT) exist. However, existing approaches are still complex and require educated users [4]. Moreover, most current WoT solutions are ill-suited for cross-application support

on mobile devices and do not support strong protection of key material by means of hardware security modules.

In this work, we propose Sea of Lights (SoL)¹, a practical lightweight scheme for bootstrapping device-to-device security and for wirelessly spreading it to enable secure distributed self-organized networks. SoL is designed to complement existing self-organized network solutions by providing a lightweight and agile solution for decentralized authentication, key management, and trust management.

Any third party mobile app can utilize the services of SoL, which offers an interface to access its security services. The security configuration can be performed on per-app granularity. Public/private key pairs generated by a third party application (=sub-keys) can be authenticated with SoL. Hence, our framework registers the public key of a sub-key, signs it, issues a certificate and is further responsible for its distribution. To this end, SoL comprises two layers. (1) The *Trust Management Layer*, which manages all operations related with trust relations. It is in charge of bootstrapping on demand, and of creating and maintaining trust relations. (2) The *Key Management Layer*, which is responsible for generating key material and managing the access to these keys in a secure fashion. This layer cares for, e.g., choosing an appropriate key storage according to the hardware capabilities and the available secure elements on the mobile devices.

In summary, the contributions of this paper are the following:

- We introduce SoL, a cross-application framework for bootstrapping security in device-to-device communication settings.
- As part of SoL, we design and implement an automatic and decentralized key and trust management solution for mobile devices. It adapts to the hardware capabilities of the host device and is able to utilize hardware security solutions to further improve the security of the underlying keys.
- We evaluate the performance of SoL on real devices to demonstrate and test its feasibility in practice and further assess its key performance features by means of simulation.

This paper is organized as follows. In Section II we summarize related work. In Section III, we concisely introduce

¹The name is inspired by silent protests such as candlelight vigils, where light is spread among the candles of a large group of people, effectively forming a ‘sea of lights’.

our motivating scenario, the system model, and the adversary model. Section IV describes the design and introduces the architectural concepts of the SoL framework, while Section V provides implementation details. We present the results of the evaluation of SoL in Section VI, covering both, measurements from our Android implementation as well as simulation results. Finally, in Section VII, we critically discuss implementation and performance issues, point to future work, and draw a conclusion.

II. RELATED WORK

Existing work in the field of security in decentralized networks focuses mainly on the communication, providing secure routing protocols solutions [5], [6], improving the fairness of the users in the network [7], increasing the robustness of such networks by detecting corrupt nodes [8], [9], etc. Our work aims at proposing a solution to bootstrap security services, while integrating a scheme for authentication and key and trust management in a decentralized fashion. Quite a number of studies focusing on trust establishment [10] and key exchange using mobile devices have also been extensively analyzed [11], [12]. Furthermore, there are also some proposals using existing security hardware on smartphones [13] and how these can be exploited to create different security levels. However, most of them either assume the existence of servers or lack an evaluation in both simulation and real devices. Our work differs from the aforementioned solutions in that we propose, inspired by the approaches based on WoT model [14], [15], a more bare-bones implementation of decentralized authentication to make it more practical. Yet, our scheme offers cross-application support and easy integration into existing apps. In our proposed scheme, each entity creates its public/private key and—after handshaking—issues certificates for its neighbors. In addition, we consider the use of secure elements to provide a secure mechanism of key management locally in the devices. Finally, we investigate the performance by means of simulation, and we test our implementation on real devices.

III. SCENARIO AND SYSTEM MODEL

In this section, we introduce the emergency response communication scenario that serves as a running example throughout the paper. We further present the general SoL system model as well as the adversary model.

A. Emergency Response Communication Scenario

Recent disasters [1]–[3] severely affected the information and communication capabilities of the population by damaging infrastructure, making communication systems unavailable, or knocking out the power. As a result, millions of people in need for help were literally left ‘in the dark’, without ready-to-use access to backup power and striped even of basic means of communication. A number of technical solutions exist to enable civilian volunteers and rescuers to build self-organized distributed wireless networks, thus enabling the population to communicate without relying on a centralized

infrastructure. These networks build on user participation and leverage mobile ad hoc networks (MANET) or delay-tolerant networks (DTN) technology to facilitate message routing/forwarding/spreading in the affected area. A typical assumption in emergency scenarios is that only honest users participate in establishing and running the network, and existing solutions often forgo security means [16]. As a result, users with malicious intent may limit or affect the communication, thus causing serious threats on the credibility and reliability of the data. Throughout the rest of this paper, we will use the aforementioned emergency response scenario as a running example. However, this does not limit the generality of the proposed SoL framework.

B. System Model

We consider users owning mobile devices capable of direct device-to-device communication. In the following, we use the term entity to refer to the logical entity formed by an authorized user and her device. Each device is imprinted with an identity, which is unique and unchangeable. For a smartphone this could be the International Mobile Equipment Identity (IMEI), a unique device fingerprint, etc. Each device has a mechanism to discover devices in its proximity such as a one-hop neighbor discovery mechanism. Applications running on the devices are assumed to be independent from each other, i.e., each application can define its own set of security requirements. No prior trust relationships or security association between entities exist, i.e., no information or knowledge about other entities is stored on a device beforehand. Centralized infrastructure to bootstrap security is unavailable. By means of direct contact and with the users in the loop, pairwise trust relationship between entities can be established. Note that we do not assume any technology for the data routing/forwarding/spreading, since SoL operates agnostic to such mechanisms.

C. Adversary Model

We assume adversaries that can act passively or actively as insiders, i.e., our adversary is a regular user of the network. Adversary capabilities follow the Dolev-Yao assumptions [17]: the adversary is, hence, capable of active interception or modification of traffic, she can fabricate and destroy messages, but is not able to break cryptographic primitives. The key goal of SoL being to bootstrap security, i.e., to provide authentication, key management and trust management services to the users, we define the main attack goals to be to disrupt these services. In particular, this entails to impersonate other entities within the network.

IV. SEA OF LIGHTS FRAMEWORK DESIGN AND ARCHITECTURE

In this section, we highlight the design concept and architecture of the proposed SoL framework. We explain the technical realization in detail in Section V. SoL is a framework that provides cross-application security services for device-to-device communication settings. Our architecture comprises a key

management as well as a trust management component, which are managed and developed as two independent elements.

A. Decentralized Authentication

The proposed decentralized authentication is based on a simplified version of the WOT paradigm, where each mobile device generates its own public/private key pair and signs the public key of others devices. Similar to other WOT solutions, trust in SOL is determined by a trust level and a maximum certification path, the so-called *degree*. We define the trust levels as follows.

- **Ultimate (U)** for the owner,
- **Trusted (T)** for direct trust relations, where an object signed directly by **U** is trusted,
- **Known (K)** for transitive relations (second degree and further), where a object signed directly by **(T)** is known, or an object signed by **n-K** is defined as known. **n** represents the minimum number of known signatures required to validate an unknown signature. Additionally, the degree defines if a transitive relation can still be considered known. We do not set a fixed value of **n** and *degree*, but allow for user configuration. This enables to tune the scalability of the system to different use cases.

In our approach, trust management is carried out in two main steps, each implemented using a dedicated protocol. The *handshake protocol* covers the bootstrapping and establishment of mutual trust, and the *synchronization protocol* manages the unidirectional synchronization of the local trust repository.

We denote a public key as pk , e.g., $pk[a]$ is the public key of **Device A**. Note that a certificate is represented as $signature[issuer, subject]$, where *subject* is the device whose public key was signed, and *issuer* is the device who signed the public key of the *subject*, e.g., $signature[a, b]$ represents the certificate of $pk[b]$ issued by **Device A**.

1) *Handshake Protocol*: Figure 1 illustrates the data flow between two devices performing the handshake protocol. It consists in the exchange of public keys and signatures between devices in proximity, in order to establish a direct trust relationship between two devices. The key verification is performed using existing Out-of-Band (OoB) verification methods (see V-A). Since comparing all bytes of public keys can be tedious and susceptible to errors, we use a short representation of these called *fingerprint*. In our approach, a fingerprint is the cryptographic hash value of any given public key. Once the OoB fingerprint verification is successful, the devices generate a certificate and assign a trust level according to the process mentioned previously. These certificates are then exchanged between devices and stored locally in their repositories.

2) *Synchronization Protocol*: Once a trust relationship has been bootstrapped, the devices can obtain information about the transitive trust relations. Figure 2 clarifies this process. The synchronization of the trust repository between two devices operates as follows: **Device A** requests from **Device B** information related to a set of known devices, **Device B** responds with the public keys and signatures related to the

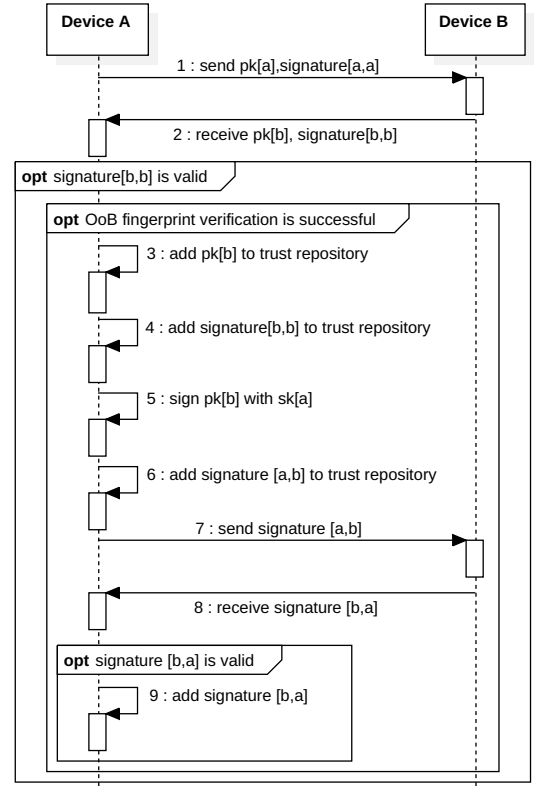


Fig. 1. Bootstrapping trust process from **Device A**'s perspective

queried devices. Finally, **Device A** merges the information into its local trust repository.

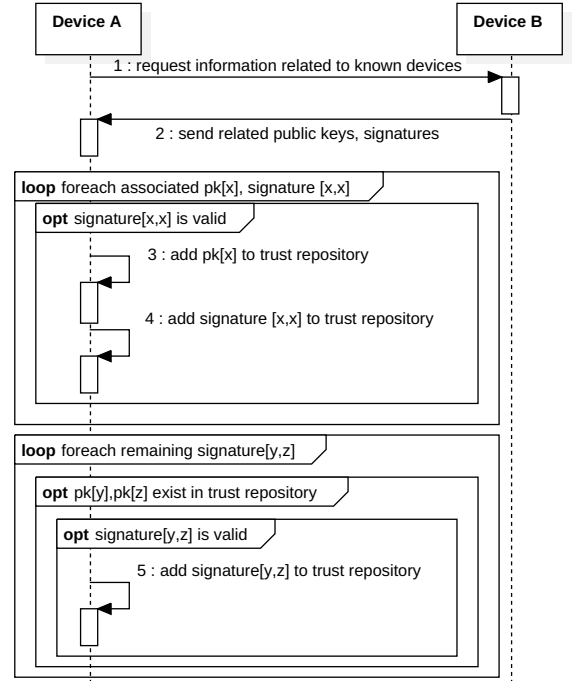


Fig. 2. Unidirectional synchronization process from **Device A**'s perspective

B. Key Management

The key management component is responsible for the management and protection of the private authentication keys from misuse and key extraction. The proposed solution is designed and implemented as a flexible solution, where the methods for the key management can be hardware- or software-based solutions. It depends on the methods supported by the devices, e.g., keys can be stored in external Near-Field Communication (NFC) tokens, TEE-based storage as AndroidKeyStore, etc. Note that irrespective of the method, the selected storage method requires a PIN, password or an additional unlock mechanism. Our approach involves two group of keys: (1) the initial authentication key, which only aims to achieve authentication and trust between the devices; and (2) sub-keys.

Sub-keys are public/private key pairs, which can be used to provide additional security properties, e.g., confidentiality. These keys are generated by a third party app and authenticated with our framework using the initial authentication key.

C. Architecture

As shown in Fig. 3, SoL is designed as a two-layer framework, which handles both trust and key management. It resides on the application layer of the Android architecture.

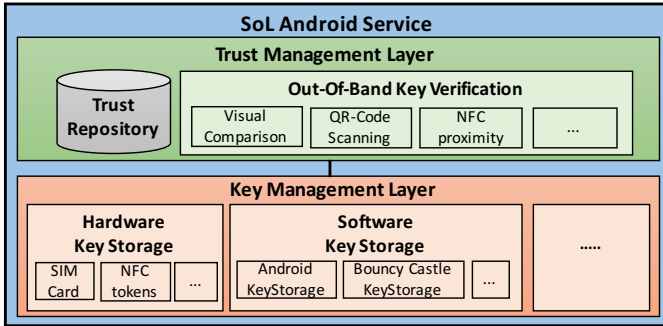


Fig. 3. The SoL architecture is designed as an Android service

1) *Trust Management Layer*: The main task of this layer is the maintenance of the trust relationships on a device. It includes the management of the trust repository, controlling the data, e.g., existing public keys, certificates, sub-key certificates, as well as checking the validity and trustworthiness of incoming data. Furthermore, this layer performs the protocols mentioned in IV-A: handshake and synchronization.

2) *Key Management Layer*: The main goal of this layer is the initialization of the authentication private key, and the support of additional operations where the initial authentication key is involved. It includes to sign keys, to issue certificates, etc. Moreover, this layer controls whether a storage is unlocked before performing any cryptographic operation. Whenever the storage is locked, a user interaction is needed for unlocking it, e.g., by introducing a password, lock pattern, PIN, etc. This layer also offers an interface to the upper Trust Management Layer.

V. IMPLEMENTATION

We developed the SoL framework for the Android platform and implemented it as a bound service² (=SoL *service*) by using Android Interface Definition Language³. The SoL *service* is encapsulated inside a standalone Android application running in its own process. It implements the trust and key management layers defined in IV-C. In the following, we discuss the most important implementation details for all SoL components.

A. Trust Management Layer

We use **fingerprints** and **key IDs** to identify a longer key with a short representation. The fingerprint is calculated using SHA-256, and the key ID is derived from the 64 LSBs of the public key. We create a directory that contains all data concerning trust relations for each known device (=subjects). The directory's name is the hexadecimal representation of the fingerprint for a subject public key. The directory contains the subject public key, signatures over the subject public key, as well as all sub-keys and their respective certificates attached to the subject. All these data are serialized and stored persistently as Base64-encoded files. The generated files are located in the application private directory.

This layer is also responsible for the key verification. After successful completion of the handshake protocol, the key exchanged in this protocol needs to be authenticated. Our framework allows for easy extensibility by facilitating the new implementation, extension, or replacement of key authentication modules. Currently, we have implemented the following existing authentication mechanisms:

- **Visual comparison**: The remote and the own fingerprint are color-coded and displayed to the users as shown in Fig. 4.
- **Scanning a QR code**: Both fingerprints are encoded in Base64 and encapsulated in a QR code as shown in Fig. 4. The user is required to scan the remote QR code. The scanning is performed using the ZXing (Zebra Crossing) project [18].
- **Using NFC technology**: When the devices are in close proximity, they exchange the fingerprints automatically.

B. Key Management Layer

The Key Management Layer implements the necessary mechanisms to provide a modular and flexible key storage solution. Thus, this layer is split into the following sub-modules:

- **SoftwareKeyManager**: This sub-module supplies a software-based keystore solution developed for Android versions before 4.3. The keystore is based on the Bouncy Castle library. The files generated in this module are protected using a user-provided PIN or pass-phrase.
- **AndroidKeyManager**: The AndroidKeyManager is the solution for Android versions from 4.3 on. This module

²<https://developer.android.com/guide/components/bound-services.html>

³<https://developer.android.com/guide/components/aidl.html>

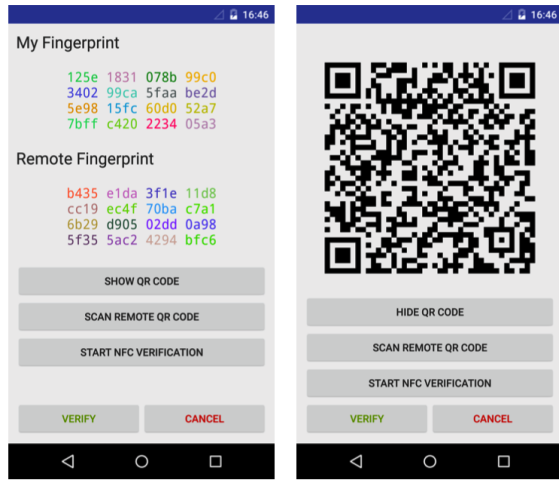


Fig. 4. Example of two key verification methods implemented in SoL

utilizes the official Android API keystore, which introduces an application private credential storage concept, but also (if available) enables additional security by offering support for hardware-based solutions.

- **HardwareKeyManager:** The HardwareKeyManager represents an additional abstraction layer for the key storage. It is the basis module for all hardware-based solutions, as all these solutions employ a similar protocol based on certain Application Protocol Data Unit (APDU) commands to communicate with Java Card Applets. Our current implementation covers three hardware-based solutions, which include the key storage and also perform the required signature operations: (1) *SmartcardManager* handles the communication with NFC smart cards. (2) *SeekManager* supports the connection to available readers, e.g., UICC. The Seek Manager manages the communication with the existing SEEK for Android framework [19]. (3) *YubiKeyManager* permits the communication with YubiKey NEO hardware tokens [20].

C. Solution-dependent Settings

In our implementation, we abstract the network layer tasks, that is, neighbor discovery and data transmission. This abstraction allows to replace the ad-hoc communication with another technology at any time. In our proof-of-concept, we used Wi-Fi Direct as the ad-hoc communication technology. Moreover, we prioritize the selection of the key manager according to the solutions supported by the device.

1) *Configurable Properties:* We define several properties as configurable:

- **maxdegree:** defines the maximum number of transitive relations that can still be considered valid.
- **numknown:** fixes the number of required known signatures to validate an unknown signature.
- **maxsubkeys:** determines the maximum number of sub-keys that an application can register.
- **signaturealgorithm:** represents the selected signature algorithm, e.g., RSA or ECDSA.

2) *Choosing the most suitable key manager:* Our selection prioritizes the hardware-based solutions. During the initialization of the SoL service, it checks whether any reader is available. If it exists, the *SeekManager* is chosen. If it does not exist, we ask the user if she wants to utilize other supported hardware-based method, e.g., *SmartcardKeyManager* or *YubiKeyManager*. Otherwise, we examine the running Android version and automatically select the suitable software-based module.

D. Integration SoL Android service by third party applications

We provide a proxy library (=SoL library) that allows an app to communicate with the SoL service in a simple and direct fashion.

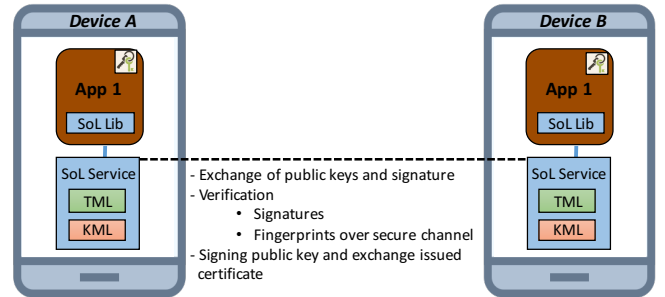


Fig. 5. SoL Android service performing the handshake protocol

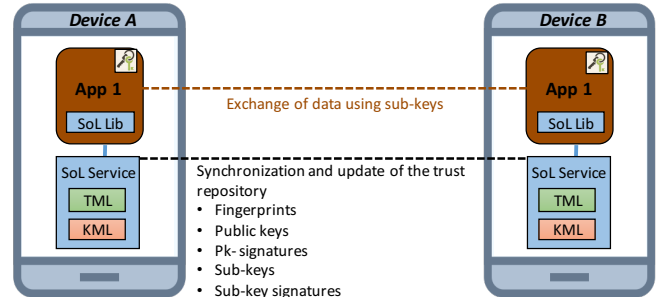


Fig. 6. SoL Android service performing the synchronization protocol

The main tasks of this proxy library include:

- Validate the installation, availability and successful initialization of the SoL service.
- Trigger the service to start the handshake protocol.
- Check existing trust relationships with a neighbor.
- Retrieve additional information about a neighbor.
- Request and register app specific sub-key certificates.
- Return the sub-keys associated to a specific fingerprint.

To clarify the use of our SoL framework, we assume the following scenario. Let **App 1** be a chat app that wants to provide a secure communication between devices in a decentralized manner. For doing so, it creates a sub-key using an asymmetric algorithm scheme. To benefit from our framework, **App 1** uses the *SoL library* to register the generated public key in our framework. Hence, the SoL service signs the key and issues a certificate. Finally, the certificate and the public key are stored in our local trust repository. Then, the SoL

service takes care of the distribution and synchronization of the public key as well as its certificate. It implies that multiple apps running on a smartphone neither need to build nor to maintain their own trust repository. Figures 5 and 6 show the data flow between devices in the aforementioned scenario.

VI. EVALUATION

TABLE I
EXPERIMENT SETTINGS

Scenario	Dimensions $w \times h$	3000 x 3000 [m]
	Simulation duration	12 [h], i.e., 720 [min] or 43200[s]
	Number of nodes	120
	Experiment	6 (3 trust degree x 2 signature algorithm)
	Runs	5 per experiment
Mobility	Model	RWP
	Speed	0.5, 1.5 [m/s]
Routing	Algorithm	DirectContact
	Buffer size	20 [MB]
Communication	Transmit speed	2 [Mbps]
	Transmission range	10 [m]
Trust management	Maximum trust degree	1 (direct) - 3
Key management	Number of Sub-keys	3
	Size per Sub-key	4096 bit
	Signature algorithm	RSA (2048 bit) ECDSA (256 bit)

The goal of the evaluation is twofold. First, we investigate the performance and scalability aspects of the trust management and the key management layer by means of simulation using the Opportunistic Network Simulator (ONE) [21]. Second we demonstrate and test our implementation on real devices to show its feasibility and test the computational performance of the key management part (see also [22] for existing performance studies of the employed algorithms).

A. Trust Management

Detailed simulation settings for the ONE are provided in Table I. We analyze four evaluation metrics for this layer: the propagation of trust relations, memory consumption, bandwidth consumption and computational overhead. The maximum degree of transitive trust relations varies from [1,3]. Each node starts with a maximum of 3 sub-keys. Each experiment runs for 12 hours (720 minutes) and nodes exchange their data every 10 seconds, if in proximity. The plots show averages over 6 different experiments: one per transitivity degree (degrees 1 to 3) using 2 possible signature algorithms (RSA or ECDSA). Each run is seeded with numbers from the interval [1,5], resulting in a total 30 runs. We use BouncyCastle JCA for signing and key generation operations. We show average values and omit the confidence intervals, which are sufficiently small and would hamper readability.

1) *Trust relationships*: Figure 7 shows the number of direct trust relationships as well as the implicit relationships. While implicit relationships increase exponentially, the direct trust indicates a linear property. The number of direct trust relationships remains almost the same and it does not depend on the maximum certification path. But rather, it varies according to the number of performed handshakes between devices.

2) *Memory and bandwidth consumption*: For this experiment, we determine the file size required for public keys, signatures, sub-keys and sub-keys signatures in the repository.

TABLE II
PROOF-OF-CONCEPT SETTINGS

Scenario	Signature algorithm	RSA (2048 bit) ECDSA (256 bit)
	Devices	1 ThinkPad X220, 8GB RAM, Ubuntu 64 bit 1 Google Nexus 5, Android version 6.0 1 YubiKey NEO (NFC token)
Procedure	I. Key Generation	2 key-pairs ($kp1, kp2$) 1 invalid signature
	II. Issue signatures	1000 by $kp1$ 200 by $kp2$
	III. Verifications	1000 (valid) 200 (invalid)

As depicted in Figure 8, the memory consumption is directly influenced by the selected trust degree as well as the signature algorithm.

First, our results confirm the existing findings regarding both algorithms: RSA exhibits higher memory usage than ECDSA for the generation of the primitives public key, sub-keys and the signatures. As the number of collecting signatures of known devices increases with each neighbor encounter, the storage space in the repository is mainly occupied by signatures. This in turn implies a considerable memory overhead, thus each node collects and stores signatures according to the trust degree selected.

Figures 9 and 10 show the bandwidth overhead and usage required for the handshake and the synchronization phase. Although, the bandwidth usage is constant during the handshake protocol, it increases rapidly in the synchronization phase depending on the selected maximum degree of trust relations.

Furthermore, if we split the data transferred during the synchronization phase into **query** and **response** operations, we notice that query operations account for the overwhelming part of usage bandwidth during the synchronization, which further increases with increasing trust degree. This is a very important result: on one hand, a trust degree higher than one is important to scale up the WOT faster, on the other hand such higher degree burdens the network and, thus, may impact the expansion of the WOT due to overload situations. In Section VII we suggest possible optimization mechanisms to minimize this issue.

3) *Computational Overhead*: We analyze the computational overhead of our solution based on numbers of operations realized during the simulation. Because these number of operations is the same for both signature algorithms, we do not separate the results into ECDSA and RSA. As shown in Figure 11, verification represents the most significant operation performance wise. Its growth is exponential and directly associated to the trust degree.

B. Key Management Layer

A proof-of-concept of SOL was implemented on Android-based smartphones to demonstrate the feasibility of our frame-

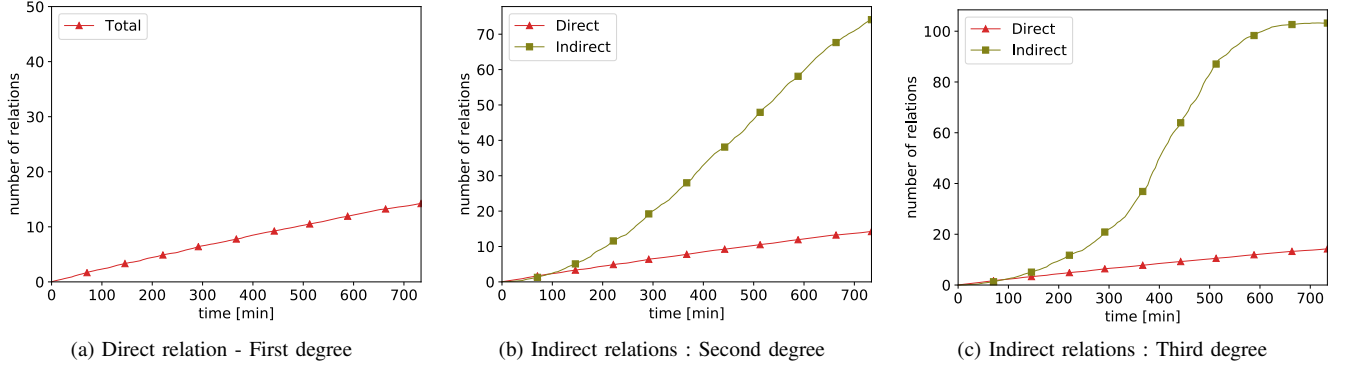


Fig. 7. Propagation of direct and indirect trust in the network

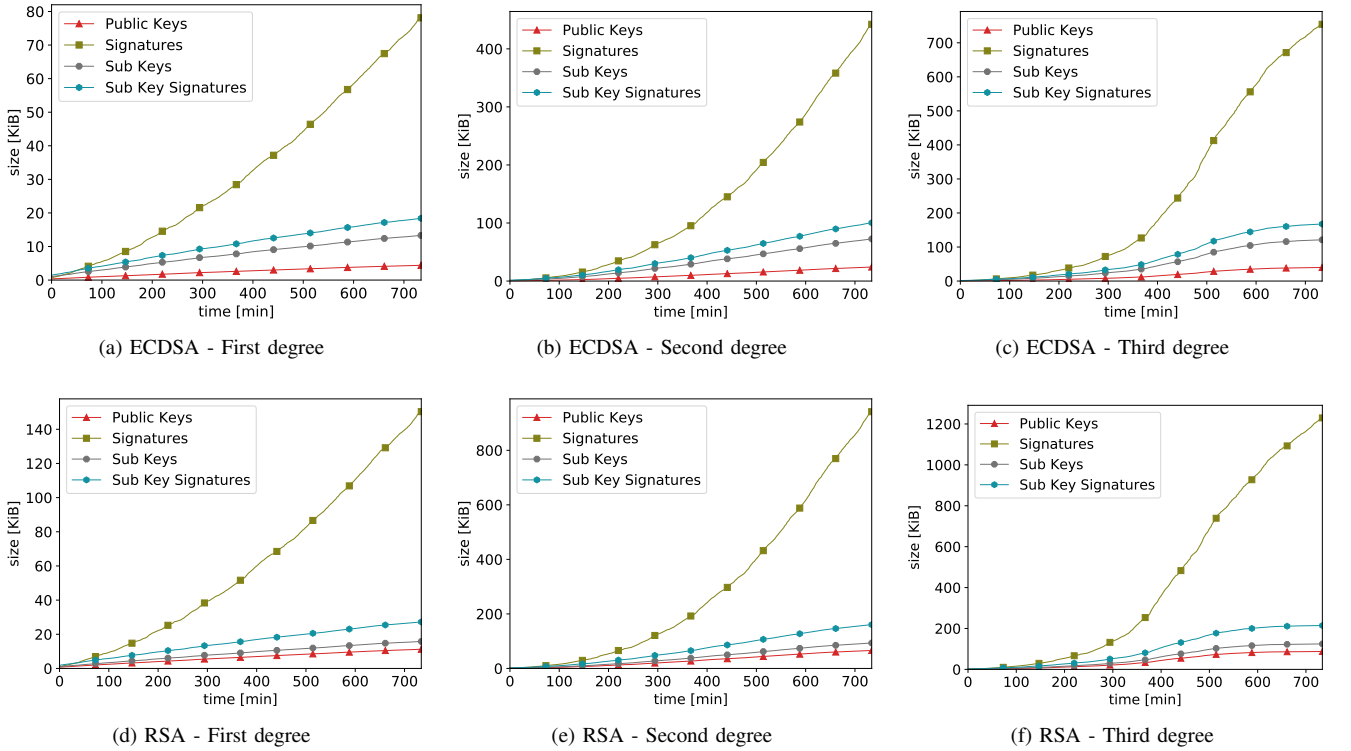


Fig. 8. Comparison of memory consumption using different cryptographic algorithms

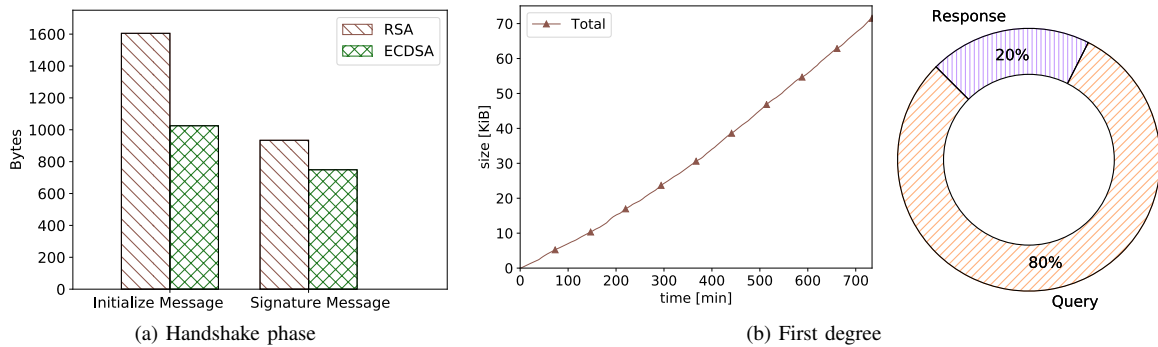


Fig. 9. Bandwidth usage during: (a) handshake phase, (b) synchronization phase: total (left) - query and response operations (right)

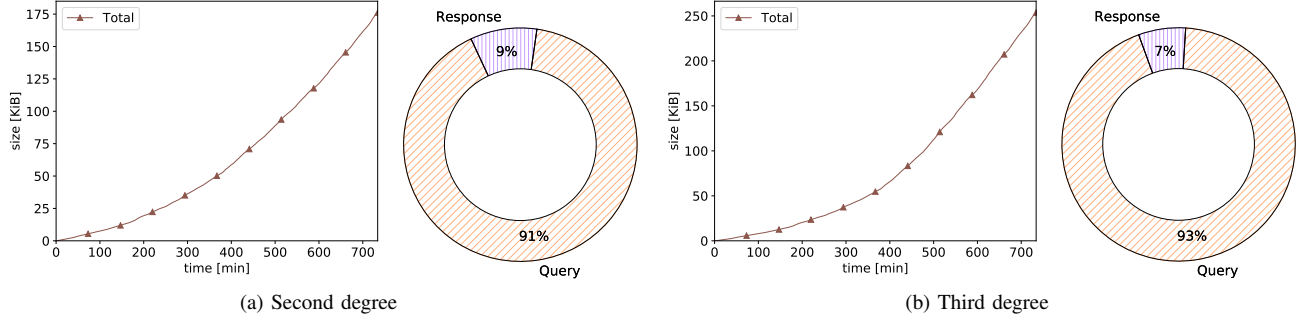


Fig. 10. Bandwidth usage during synchronization phase: total (time plots) and query and response operations (pie plots)

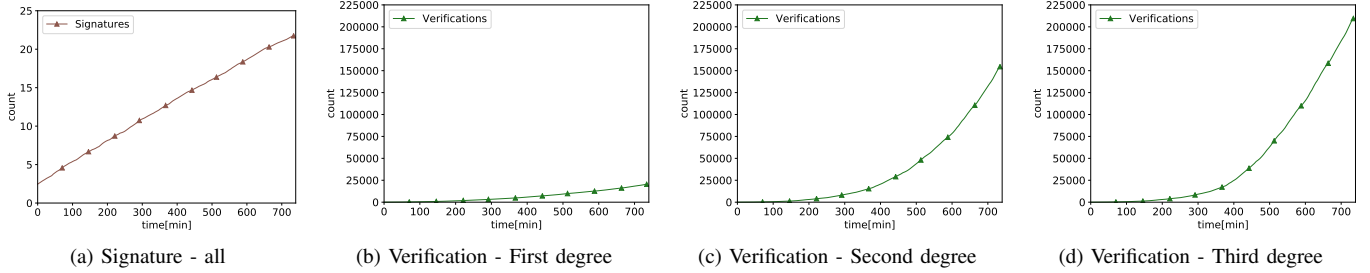


Fig. 11. Computational overhead: total numbers of signing and verification operations during the simulation

work on real devices.

The evaluation of this layer covers the two supported cryptographic algorithm, and compares both in terms of efficiency and performance. Additionally, we compare hardware- and software-based key storage mechanisms. Detailed experiment settings are provided in Table II. The hardware-based storage experiment is performed with a smartphone and a NFC token.

Additionally, we use a laptop for the software-based storage experiment. We repeat the experiment 15 times.

Each iteration takes place as follows:

- 1) First, two key pairs are generated: *kp1* that is considered as valid, and *kp2* which issues invalid signatures. It means a signature is valid only if it was issued by *kp1*.
- 2) After the key generation, *kp1* issues 1000 signatures, and *kp2* issues 200 signatures.
- 3) Finally, the issued signatures are verified using *kp1*.

Figure 12 shows the performance of RSA and ECDSA executing cryptographic operations: generation, signing, and verification. ECDSA has a better performance regarding key generation and issuing signatures. RSA, however, is more efficient for verifying signatures. Note that the hardware-based approach offers a poor performance for signing operations, since the processor embedded in the NFC token is relatively slow. Notwithstanding, this result depends directly on the capabilities in terms of cryptographic operations supported by the token. For example, if a token has a dedicated cryptographic co-processor for such operations, it will be faster compared with another one who performs cryptographic operations on

the regular micro-processor.

VII. DISCUSSION AND CONCLUSION

This paper presented SOL, a practical framework to bootstrap security for device-to-device settings. SOL is designed and implemented as an Android service. SOL uses asymmetric cryptographic algorithms: RSA and ECDSA. It also implements a simplified version of the WOT paradigm. It features a Trust Management Layer and a Key Management Layer. The former manages all operations and methods related with the trust relations: bootstrapping, maintaining and synchronization. It deals also with the OoB key verification. The latter performs all operations concerning to the underlying keys: initialization, generation and management. It supports both software and hardware-based key storage solutions. Furthermore, third party apps which utilize our library can benefit from our framework by offering an authenticated and secure communication. To this end, they can create and register the application-specific sub-keys in our service. Finally, the implementation of a proof-of-concept demonstrates the feasibility of our solution on real devices. Simulation results confirm the trade-off between trust transitivity and synchronization overhead: transitive trust facilitates a much faster coverage, while direct trust is conserving bandwidth, but limits trust coverage. We make our source code publicly available [23].

We foresee a number of performance improvements. Introducing the concept of a timeout interval together with a register to remember users and timestamp of previous encounters can help saving bandwidth. Then, a new synchronization with a specific user is only allowed after the timeout has expired.

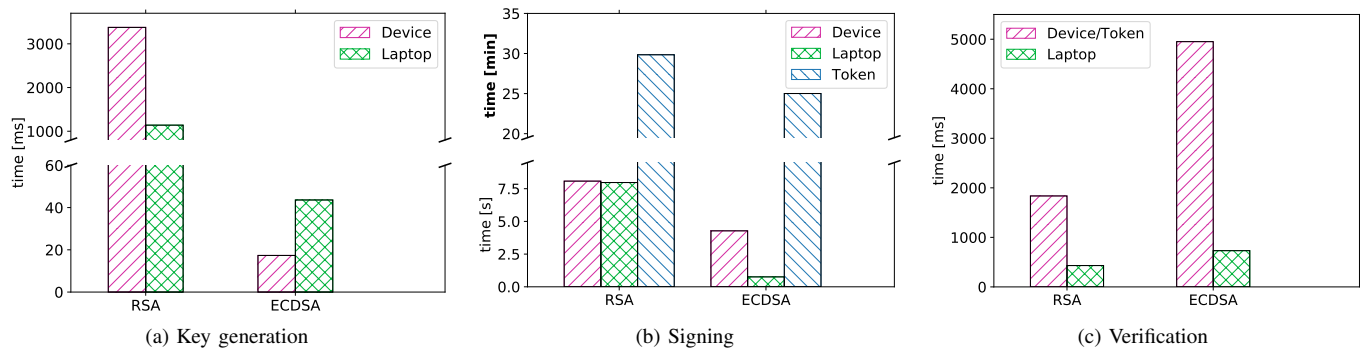


Fig. 12. Performance comparison between RSA and ECDSA executing cryptographic operations

Furthermore, we can also employ bloom filters for checking already known or trusted users. Thus, it constrains the query part from the synchronization phase.

Finally, even though our scheme builds on direct physical interaction of users and the trust level of transitive relations can be configured. I.e. there is a certain control on how far information generated by malicious devices will spread into the network. There are still open issues that are only partially covered or not covered by our solution, for example, an important aspect remains the implementation of methods focused on the revocation of compromised keys in decentralized networks. Indeed, key revocation plays an important role for keeping security in a network as a compromised key can affect the trust of the system partially or totally. However, as summarized in [24], there is no one-for-all key revocation scheme which deals with all the security issues and requirements of such networks. For example, many of them assume the existence of a central authority or a prior known of the network topology, which is not always possible especially in mobile real-world systems, where devices can join and leave the network arbitrarily. Several schemes rely on the information provided by the devices in the network in order to deal with misbehaving users. However, this can also be used by malicious users to affect the network and to disable legitimate devices.

ACKNOWLEDGMENT

This work has been funded by the German Federal Ministry for Education and Research (BMBF) within the SMARTER project.

REFERENCES

- [1] The Times of India, "Nepal-India earthquake." [Online]. Available: <http://timesofindia.indiatimes.com/>
- [2] C. Ray Sanchez, "Power outages: Post-Irma recovery includes turning on the lights." [Online]. Available: <http://edition.cnn.com/>
- [3] L. A. Times, "Censors in China keep mainlanders in dark about Hong Kong protests." [Online]. Available: <http://www.latimes.com/>
- [4] A. Whitten and J. D. Tygar, "Why Johnny can't encrypt: A usability evaluation of PGP 5.0." in *USENIX Security Symposium*, vol. 348, 1999.
- [5] M. Schmittner, "Scalable and secure multicast routing for mobile ad-hoc networks," Master's thesis, Technische Universität, 2014.
- [6] L. Li, X. Zhong, and Y. Qin, "A secure routing based on social trust in opportunistic networks," in *IEEE ICCS*, 2016.
- [7] S. Buchegger and J.-Y. Le Boudec, "Nodes bearing grudges: Towards routing security, fairness, and robustness in mobile ad hoc networks," in *IEEE EMPDP*, 2002, pp. 403–410.
- [8] E. M. Shakshuki, N. Kang, and T. R. Sheltami, "Eaacka secure intrusion-detection system for manets," *IEEE TIE*, vol. 60, no. 3, 2013.
- [9] A. Nadeem and M. P. Howarth, "An intrusion detection & adaptive response mechanism for manets," *Ad Hoc Networks*, vol. 13, 2014.
- [10] J. Seedorf, D. Kutscher, and F. Schneider, "Decentralised binding of self-certifying names to real-world identities for assessment of third-party messages in fragmented mobile networks," in *IEEE INFOCOM WKSHPS*, 2014.
- [11] M. Farb, Y.-H. Lin, T. H.-J. Kim, J. McCune, and A. Perrig, "Safeslinger: easy-to-use and secure public-key exchange," in *ACM MobiCom*, 2013.
- [12] W. Shen, W. Hong, X. Cao, B. Yin, D. M. Shila, and Y. Cheng, "Secure key establishment for device-to-device communications," in *IEEE GLOBECOM*, 2014.
- [13] C. Busold, A. Taha, C. Wachsmann, A. Dmitrienko, H. Seudié, M. Sobhani, and A.-R. Sadeghi, "Smart keys for cyber-cars: secure smartphone-based NFC-enabled car immobilizer," in *ACM CODASPY*, 2013.
- [14] S. Capkun, J.-P. Hubaux, and L. Buttyan, "Mobility helps peer-to-peer security," *IEEE TMC*, vol. 5, no. 1, 2006.
- [15] M. Cagalj, S. Capkun, and J.-P. Hubaux, "Key agreement in peer-to-peer wireless networks," *Proc. IEEE*, vol. 94, no. 2, 2006.
- [16] F. Alvarez, M. Hollick, and P. Gardner-Stephen, "Maintaining both availability and integrity of communications: Challenges and guidelines for data security and privacy during disasters and crises," in *IEEE GHTC*, 2016.
- [17] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE TIT*, vol. 29, no. 2, 1983.
- [18] "ZXing Project." [Online]. Available: <https://github.com/zxing/zxing>
- [19] "SEEK for Android." [Online]. Available: <https://seek-for-android.github.io/>
- [20] "YubiKey." [Online]. Available: <https://www.yubico.com/>
- [21] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE simulator for DTN protocol evaluation," in *ACM Simutools*, 2009, p. 55.
- [22] R. Sinha, H. K. Srivastava, and S. Gupta, "Performance based comparison study of RSA and elliptic curve cryptography," *IJSE*, vol. 4, no. 5, 2013.
- [23] "SOL source code." [Online]. Available: <https://seemoo.de/sol>
- [24] M. Ge, K.-K. R. Choo, H. Wu, and Y. Yu, "Survey on key revocation mechanisms in wireless sensor networks," *JNCA*, vol. 63, pp. 24–38, 2016.