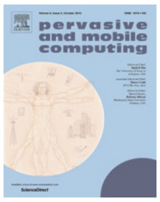
Pervasive and Mobile Computing 32 (2016) 26–34

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc



Decoupling data-at-rest encryption and smartphone locking with wearable devices

Ildar Muslukhov*, San-Tsai Sun, Primal Wijesekera, Yazan Boshmaf, Konstantin Beznosov

The University of British Columbia, 4085-4224 Main Mall, Vancouver, BC, Canada

ARTICLE INFO

Article history:
Available online 5 July 2016

Keywords:
Smartphone loss and theft
Data-at-rest encryption
Smartphone locking
Wearable devices
Encryption keys management

ABSTRACT

Smartphones store sensitive and confidential data, e.g., business related documents or emails. If a smartphone is stolen, such data are at risk of disclosure. To mitigate this risk, modern smartphones allow users to enable data encryption, which uses a locking password to protect the data encryption key. Unfortunately, users either do not lock their devices at all, due to usability issues, or use weak and easy to guess 4-digit PINs. This makes the current approach of protecting confidential data-at-rest ineffective against password guessing attackers. To address this problem we design, implement and evaluate the Sidekick system — a system that uses a wearable device to decouple data encryption and smartphone locking. Evaluation of the Sidekick system revealed that the proposal can run on an 8-bit System-on-Chip, uses only 4 Kb/20 Kb of RAM/ROM, allows data encryption key fetching in less than two seconds, while lasting for more than a year on a single coin-cell battery.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

With the number of active smartphone users approaching two billion, these devices have become indispensable for many users and companies. Users often store sensitive and confidential data, e.g., photos, videos, email messages, and saved credentials. Unfortunately, only 50% of smartphone users choose to protect their data by enabling password-based encryption (PBE) [1,2].

At the same time the wide adoption of the bring your own device (BYOD) policies in the organizations, i.e., a policy that allows employees to use their own smartphones for business related purposes, creates a risk of corporate data disclosure [3]. Indeed, recent reports show that **smartphone theft** and **confidential information disclosure** have significant impact on companies. In the US alone, every tenth smartphone owner has his smartphone stolen at least once [4]. More than 30% of all street robberies involve a smartphone theft [5], and the number of stolen smartphones has doubled in 2014, reaching 3.1 million devices [6]. In addition, 46% of companies from North America and Europe stated that a theft of a data bearing device, such as a smartphone, was the key factor in the data breaches they experienced [3].

The current approach to reduce **the risk of sensitive information disclosure** on smartphones is based on data encryption. Users can enable PBE of data by setting up an encryption password. For usability reasons, in Android and iOS

* Corresponding author.
E-mail addresses: ildarm@ece.ubc.ca (I. Muslukhov), santsais@ece.ubc.ca (S.-T. Sun), primal@ece.ubc.ca (P. Wijesekera), boshmaf@ece.ubc.ca (Y. Boshmaf), beznosov@ece.ubc.ca (K. Beznosov).

http://dx.doi.org/10.1016/j.pmcj.2016.06.016
1574-1192/© 2016 Elsevier B.V. All rights reserved.

the same password is used for both PBE and smartphone locking/unlocking. The operating system (OS) then transparently encrypts/decrypts data during write/read operations with a randomly generated data encryption key (DEK).[1] For protection, the DEK is encrypted with a key encryption key (KEK). The encrypted DEK is then stored on the mobile device, while the KEK is not. Android and iOS derive the KEK from user authentication secret (e.g., a PIN or a password) by using a Key Derivation Function (KDF), such as PBKDF2 (Password Based Key Derivation Function 2) or *scrypt*. The security encrypted data fully depends on the strength of the authentication secret users choose to lock their smartphones.

Unfortunately, 9 out of 10 users that lock their devices tend to choose weak authentication secrets, such as 4-digit PINs, citing usability issues as the key factors in that decision [2,7]. PINs not only can be easily eavesdropped [8], they can also be guessed in a matter of minutes [9,10]. While the evaluation results of novel and usable authentication methods for smartphones suggest that it is possible to mitigate certain threats, e.g., shoulder surfing, it is still challenging to increase the entropy of the authentication secrets [11]. In an attempt to ease the burden of typing an unlocking password many times a day, Touch Id, a biometric authentication sensor, was introduced to Apple's iPhones and iPads. Although, this sensor made it much more practical for users to employ harder-to-guess passwords, recent research showed that even with Touch Id sensor enabled, users still prefer to use weak 4-digit PINs [7].

All of the above suggests that the current data-at-rest protection is ineffective against password guessing attackers. That is, while organizations want their employees to lock their devices and use harder to guess passwords, users fail to comply and would rather disable smartphone locking all-together if forced to employ *stronger* passwords with currently available authentication methods. Even more, with the wide adoption of the BYOD, organizations are unable to enforce these requirements on to their employees, since IT departments do not have control over the smartphones in use. This makes the task of securing data-at-rest on smartphones against the risk of disclosure a hard and challenging problem for commercial and governmental organizations. That is why in this paper we aim to answer the following research question *"How can we design a practical, readily deployable, efficient and effective system that mitigates password searching attackers who aim to decrypt confidential data-at-rest while not requiring smartphone users to use stronger passwords?"*

Our Approach and Key Results. To address the problem of data-at-rest security for companies and organizations we propose to use wearable devices to store randomly generated KEKs. Proliferation of wearable devices, e.g., smartwatches or fitness trackers, makes such a proposal intriguing. First, such an approach can work with a limited connection to the Internet, which gives it a better mobility. Second, using wearable devices allows us to decouple user authentication for smartphone unlocking and data-at-rest encryption by storing randomly generated KEKs on a wearable device. Third, by using wearable devices in a personal area network (PAN) we can use proximity measures (i.e., signal strength) to implement a cryptographic "lock-down" for confidential data, which works as follows. If a wearable device gets disconnected or the distance surpasses a certain threshold, all KEKs and DEKs in smartphone's RAM are wiped out and all corresponding applications are notified about that event. KEKs and decrypted DEKs are only stored in RAM, hence, secure deletion of these keys from RAM makes it cryptographically infeasible to gain access to data they protect. Although, already decrypted data remain prone to exfiltration while kept in RAM, applications can implement reaction to the *disconnected* event and securely remove that data as well.

To test the idea of using wearable devices as KEKs storage we designed and evaluated the Sidekick system, an *effective* and *practical* system that uses an external wearable device to store randomly generated KEKs. The proposed system is ***effective*** at mitigating adversaries that mount brute-force attack on unlocking secrets, since it substantially increases the entropy of a KEK, to at least 128-bit, without relying on users to remember more complex passwords. In addition, unlike existing systems with a single KEK, Sidekick allows having multiple DEKs and KEKs, which enables the use of unique DEK-KEK pairs per file, directory or application. Such a fine grained approach coupled with a KEK wipe-out trigger based on proximity allows not only limiting the damage from a successful *cold-boot attack* (due to a reduced set of files a single DEK can decrypt), but also substantially increases required efforts for a successful *cold-boot* attack.

The evaluation of the Sidekick system revealed that the proposal is ***practical***. First, the proposed system is *cross-platform*. While we evaluated it only on iOS and Android, support for other platforms can be added trivially. Second, the Sidekick system is *backward compatible*. That is, it works on new and old devices, as long as they support Bluetooth Low Energy (BLE) — a wireless communication stack that has been included in smartphones since 2011 [12,13]. In fact, we evaluated our proposal on an iPhone 4S and a Samsung Galaxy S3, released in 2011 and 2012 respectively. Third, our proposal is *efficient*, since power consumption overheads are negligible on smartphones and tablets. Furthermore, the power consumption on a wearable device allows it to run for more than a year on a single coin-cell battery. Sidekick only requires explicit user involvement during the initial pairing of the wearable device with her smartphone, an event that happens once. In addition, minimal memory footprint (4 Kb/20 Kb of RAM/ROM) makes it possible to have Sidekick's functionality as an additional service to already available wearables. Finally, the proposal is relatively *inexpensive*, since it can be implemented on the cheapest BLE-enabled System-on-Chip (SoC). In particular, in our evaluation we used a SoC that costs under $4. At the same time the average cost of recovering from a data breach per record in 2014 was $218 and the average data breach involved 29,000 of such records [3]. A KEK can be fetched from wearable device in two seconds or less, which is faster than the most common authentication methods, i.e., PIN-codes or Draw-A-Secret [14]. Overall, we argue that the Sidekick proposal is a viable option for organizations that seek a way to protect their confidential data-at-rest in smartphones used and owned by their employees.

[1] Also called *the master key* in some sources.

2. Threat model

In this section we describe threats, risks and attackers' capabilities.

2.1. Threats and risks

The **threats** we consider in this work are smartphone loss and theft (we refer to both of them as *theft* for brevity). As a result of a theft, the device ends up in hands of an attacker who wants to gain access to data stored on the stolen device for any form of profit. Unlike existing adversarial models (e.g., see proposal by Do et al. [15]) we do not consider that the attacker is interested in obtaining data in a forensically sound manner.

Once the device is in the hands of an attacker, there is a **risk** of confidential data disclosure. If the attacker obtains confidential data then the victim might suffer losses (e.g., reputation damage or/and financial). For example, the victim might have to pay fees for recovering from identity theft or fines for leaking private customer information.

2.2. Attack

If PBE is not enabled, then the attacker can access data trivially. If, however, PBE is enabled then the attacker extracts a *bit-by-bit* image of the internal storage through his own or existing tools (e.g., [16]). The main reason for extracting the image is to bypass OS level data wipe-out, which is usually triggered after a predefined set of unsuccessful authentication attempts. For instance, in iOS user can enable device wipeout after 10 unsuccessful attempts. With the storage image in hand, the attacker mounts a password guessing attack in order to recover the KEK.

During the attack, if the stolen device uses specialized hardware for PBE, the attacker might need to run certain computations on that hardware. Such attacks are called *on-device* brute-force attacks and, in general, are significantly slower than *off-device* attacks since massive parallelization becomes unavailable. Once the KEK is recovered, it becomes trivial to recover the DEK and, consequentially, obtain access to encrypted data.

Since we are proposing using wearable devices, we ought to consider an increased attack surface. That is, we have consider attacks on the wireless stack and the implications of such attacks on the overall security of encrypted data. First, one can attempt to obtain KEKs transferred over the BLE channel. If such attack is successful, then one can decrypt DEK right away. Second, the attacker might aim to corrupt KEK during transmission over BLE. If he succeeds in corrupting the KEK, then the victim would use an incorrect KEK for DEK encryption, which later would render decryption of DEK cryptographically impossible. Even more, KEK corruption attack allows locking users' valuable data, which can be used by ransomware — a malware that encrypts data and then extorts money for the decryption key. We consider KEK corruption during transmission a significant risk to a positive user experience and system adoption.

2.3. General assumptions

We make several general assumptions in our threat model about the adversary's capabilities. First of all, we assume perfect cryptography, i.e., attackers cannot differentiate the AES cipher from a random permutation function. Considering that non-generic attacks have yet to be found for the AES cipher this assumption is sound. We also assume that there are no security bugs in the implementation of the data encryption system that would introduce a shortcut for encryption and decryption (e.g., by using a hard-coded KEK or DEK, or by using a biased and predictable random number generator).

In addition, we do not discuss or consider confidential data disclosure through OS kernel compromises for several reasons. First, having a secure OS kernel does not prevent the password guessing attack. Second, a compromised OS kernel renders any data encryption ineffective. That is, if an attacker can compromise the OS kernel, he can extract all required data by obtaining KEKs or DEKs directly from RAM.

Use of wireless communication stacks enables attackers to track users based on devices' addresses (i.e., Media Access Code). We, however, assume that tracking users is not one of the objectives of the attackers we consider. We argue that such an objective is not only unrelated to data-at-rest security, but it can also be addressed with existing methods and tools, e.g., BLE privacy feature [13]. Neither do we consider deniability of data existence as one of the main requirements. In fact, such a need can be addressed by using one of the available deniable file systems (e.g., [10]).

2.4. Crypto-attacker

The *crypto-attacker* aims to obtain confidential data-at-rest by recovering the KEK through password search. We make the following assumptions about the *crypto-attacker*'s capabilities. First, he has physical access to the victim's smartphone. Second, we assume that the *crypto-attacker* knows the design of the data encryption system and knows how the KEK is generated. Third, we assume that he can obtain a *bit-by-bit* image of the internal storage on the stolen smartphone, which allows bypassing the file system access control. Techniques that allow acquiring the raw storage image have been widely discussed in the last few years [10,17]. Fourth, if the stolen device uses special hardware for data encryption, e.g., crypto-chip in iPhones, he knows how to mount an *on-device* brute-force attack based on existing approaches [17]. Finally, we assume that the *crypto-attacker* is not capable of brute-forcing pseudo-randomly generated 128-bit KEKs.

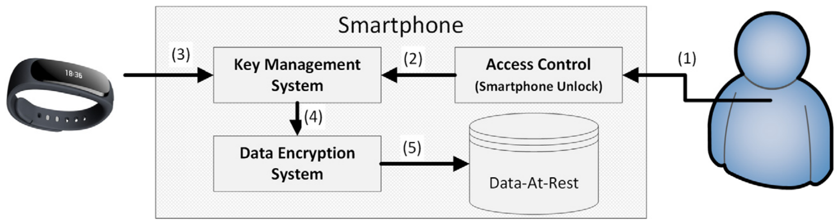


Fig. 1. A smartphone, through Access Control sub-system, requests the user to authenticate. The user provides a correct authentication secret (1), which is passed to the Key Management System (2). Key Management System at the same time receives a KEK (3) from a wearable device (e.g., a fitness tracker or smartwatch). KEK allows the corresponding DEK to be recovered, which is then passed to Data Encryption System (4) that encrypts/decrypts confidential data (5).

2.5. Network-attacker

The *network-attacker* might have several objectives. First, he might be interested in data-at-rest stored on the smartphone. In that case the attacker plans to steal the smartphone later, but first he aims to obtain all KEKs transmitted over BLE in order to eliminate the necessity to do a KEK search later. Second, the *network-attacker* might be interested in corrupting KEKs to cryptographically lock users' valuable data. The attacker can then use his knowledge of how he corrupted the KEK and request ransom payment from the victim.

To compromise the wireless channel, the *network-attacker* can use one of the two approaches. First, the attacker can focus on the wireless messages themselves by exploiting insecure protocols and gaining the ability to recover or corrupt KEKs. In particular, we assume that the *network-attacker* is able to exploit vulnerabilities reported in BLE stack thus far [18–20]. In particular, these attacks showed that recovering devices pairing key is practical, which allows the attacker decrypting and modifying any message transmitted over BLE stack. Second, if the attacker has control over an application on victim's smartphone with access to the BLE stack (cf. misbonding attack [21]), he can communicate with the wearable device directly and retrieve all required KEKs before stealing the device. In order to mount a misbonding attack one needs to get access to the BLE stack, e.g., by requesting *android.permission.BLUETOOTH* permission for an Android application. This allows the application to communicate with all Bluetooth devices that are paired and connected, including the wearable device used for KEKs storage.

3. Our approach

The third step of the flow (presented at Fig. 1) is the only part that differentiates our proposal from all existing systems. That is why in the remainder of this paper we focus on security and performance evaluation of that step. To mitigate the risks associated with the theft of a smartphone, we propose to use a wearable device in the encryption keys management sub-system. Existing systems, such as TrueCrypt and BitLocker, already support the use of physically attached devices, i.e, USB flash drives, to store encryption keys [22,23]. The use of external devices eliminates the sole dependency of KEK strength on the complexity of human-memorable secrets. In fact, it makes the use of password for data encryption completely optional. That is, an application can securely encrypt data without requiring users to set a password for data encryption to work. In that case, KEK is generated randomly and stored on the wearable device rather than derived from a human-memorable password.

Unfortunately, physically attached external devices, such as memory cards or USB flash drives, are not well suited for smartphones. First, almost all modern smartphones do not support USB flash drives or memory cards. Second, in contrast to wearables, a physical connection requires constant user attention and vigilance. That is, forgetting to unplug the external device from the smartphone destroys all security benefits that the use of a wearable device has added.

That is why we propose to use a wirelessly connected external device to store KEKs. We argue that wireless devices are better suited for KEK storage for the following reasons. First, a wirelessly connected wearable device allows removing users from the loop of obtaining required KEKs. That is, a smartphone can automatically scan the PAN and connect to the appropriate device. Second, the proliferation of wearable devices, such as smartwatches and fitness trackers, increases a set of options for a KEK storing device. Last but not least, all mobile platforms support a range of PAN stacks, e.g., BLE or NFC.

We chose to use BLE stack for prototyping for the following practical reasons. First, BLE hardware and BLE APIs are available on all platforms, while other PAN stacks (e.g., NFC) have limited support. Second, BLE allows SoC to work for months off a single coin-cell battery [24,25]. Finally, most of the released wearable devices rely on the BLE stack for communicating with smartphones (e.g., FitBit [26]). For system evaluation we decided to use CC2540 System-On-Chip (SoC) from Texas Instruments as it was the least capable SoC at the time of our experiments [27].

Of course, one can use a remote server to store KEKs. Such an approach, however, requires a reliable connection to the Internet in order to be able to fetch the KEK when needed. Using a centralized server, however, has several important disadvantages. First, the use of wide area networks (WAN) limits the scenarios where one can access sensitive data. In particular, when a user has limited connection to the Internet, e.g., while traveling. Second, in contrast with PAN stacks, WAN stacks do not provide means to measure proximity of the smartphone to the smartphone owner. That is, if a smartphone is

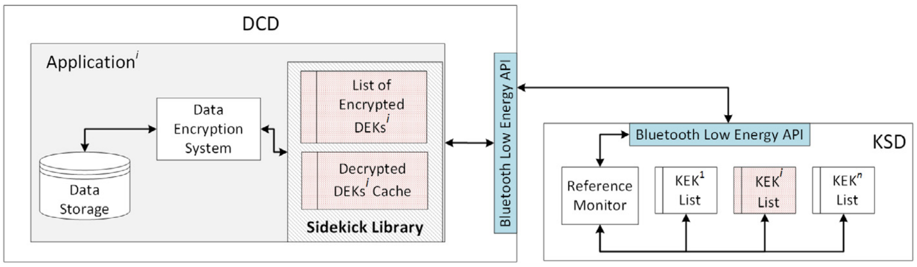


Fig. 2. High-level design of the Sidekick System. A data containing device (DCD) runs applications that link the Sidekick library. The library takes care of all communications with the KSD, e.g., storing or retrieving a KEK. Once a required KEK is retrieved, a corresponding DEK is decrypted and stored in Decrypted DEKs Cache by the Sidekick Library. The DEK is then passed to the Data Encryption System in order to encrypt/decrypt data. Each application has a separate KEK List. Reference monitor on KSD ensures that each application has access only to its own KEK List.

stolen, then an attacker still can access all KEKs stored on the central server until the theft is reported to the IT department and appropriate actions are taken. Reporting might take substantial amounts of time, considering that a user has just lost his primary communication device.

4. System design

In this section we present the design of the Sidekick system. We begin with a high-level overview of the system, then proceed with a discussion of countermeasures that mitigate the *network-attacker*. We conclude with a security analysis of the proposal in the presence of *network* and *crypto* attackers.

4.1. High level overview

The Sidekick system consists of two separate devices — (1) a data containing device (DCD), e.g., a smartphone, and (2) a key storing device (KSD), i.e., a wearable device. When a user accesses sensitive data on the DCD, the Sidekick library fetches KEKs from the KSD and recovers DEKs, which makes it possible to read/write data on DCD. The Sidekick library handles *store*, *update* and *delete* requests in a similar fashion. The block diagram is shown at Fig. 2.

Each of the four requests (Req) that the DCD sends has a corresponding response (Resp) from the KSD. For instance, when the DCD needs to store a new KEK on the KSD, it sends a ***StoreReq*** to the KSD with the new KEK in the payload. Once the KSD has processed that request, it responds with a ***StoreResp***, which contains a KEK_{ID}, a unique KEK identifier, in the payload.

To make the system readily deployable we implemented the Sidekick system on the DCD as a library, rather than as a patch for an OS. In addition, having Sidekick system as a library allows us to support multiple platforms, including closed source, and platforms' versions.

4.2. Securing communications over BLE

While the main objective of Sidekick system is to mitigate the *crypto-attacker*, we have to address the risks that arise from use of the BLE stack. In particular, a *network-attacker* can exploit one of the previously reported vulnerabilities [18–20].[2] The existence of such attacks is not surprising, since the security of BLE was compromised in order to make BLE-enabled SoC power-efficient [13]. In addition, we have to address the limitations in access control in mobile OS, which allows any application on a smartphone with access to BLE stack to communicate with any paired BLE device.

Transport Layer Security. To ensure integrity and confidentiality protection for all Sidekick's communications over BLE, i.e., to mitigate the *network-attacker* who aims to corrupt KEKs, we used Counter with CBC-MAC mode (CCM) [28], because (a) in all BLE-enabled SoC this mode must be implemented in hardware [13], and (b) it has been proven to be secure [29]. Indeed, in our benchmarking experiments we confirmed that hardware-based implementation of 128-bit AES in CCM mode is in the order of two magnitudes faster than its software counterpart (0.058 ms against 4.45 ms). Although existing BLE SoC supports only 128-bit key size for hardware implementation of the AES cipher, we consider doing a complete search over 128-bit key-space is infeasible.

Mutual Authentication. On top of the transport layer, which is the Attribute Protocol (or ATT) in the BLE stack, we use a mutual authentication protocol in order to mitigate identity spoofing attacks. We adopt a well-known and studied mutual authentication protocol based on a shared secret [30].

Pairing KSD and DCD. For mutual authentication to work, one should have a shared secret first. Unfortunately, wearable devices are usually limited in Input/Output capabilities. For instance, CC2540 SoC only has two LEDs in default circuit design.

[2] As of this writing all these attacks are still practical.

Considering such limitation, we decided to use blinking LED (BLED) approach proposed by Saxena et al. [31]. In BLED, one device generates a pseudo random key and shares it through a blinking LED, while another device, DCD in our case, uses camera and converts blinking LED into a bit stream. The security of secret shared this way is only crucial for the period of time required to agree on a new shared key during KSD and DCD pairing process, e.g., through Elliptic Curve Diffie–Hellman (ECDH). In our benchmarking experimentations, both Samsung S3 and iPhone 4S reached bandwidth of 3 bits/s in key sharing over BLED, which allows to share a small 32-bit secret in about 10 s. Although this approach sounds cumbersome, a user needs to execute it only once per device, i.e., during the initial pairing process.

Session Key Establishment. If a weak authentication secret was used during pairing, one should improve it through other known protocols, e.g., Diffie–Hellman (DH) [32]. Unfortunately, due to computational constraints and limited RAM on the wearables, the original DH protocol is not practical. That is why we choose to use ECDH protocol, based on the P128 curve.

Other Considerations. In addition to identity spoofing, we had to consider *Replay* and *Retry* attacks. To mitigate *Replay* attacks we include a *Nonce* in each message and verify that *Nonce* on the recipient side. To mitigate *Retry* attacks we use a monotonically increasing number as a *Nonce*, which allows us to detect old messages by comparing the message number of the message in question with the last message sent/received thus far.

To summarize, by proposed measures we effectively mitigated the *network-attacker* presented in Section 2.5.

5. System evaluation

Experimental Setup. To evaluate the Sidekick system we used an iPhone 4S and a Samsung S3 as the DCDs, and a CC2540 SoC [33] as the KSD. We chose to use the CC2540 model of SoC as we found it to be the least capable, performance and memory wise, BLE-enabled SoC available at the time of experiment. This makes it possible to run our proposal on all available and future released BLE-enabled SoC.

Implementation. The Sidekick system consists of several components. First, there is a reference monitor (RM) and KEK handling logic on the KSD, which we implemented as firmware for the CC2540. Second, there is a communication stack over BLE that provides a secure channel between the KSD and the DCD. Most of the logic is implemented in C, with some parts being implemented in native language for the given platforms, i.e., Objective C for iOS, or Java for Android. The memory footprint of the Sidekick system can be ignored both for smartphones and for CC2540. In particular, the system needed less than 20 Kb of ROM and 4 KB of RAM. In our implementation, we used 256-bit KEKs and DEKs, 32-bit integers as a KEK_{ID}, a single byte as operation code (*OpCode*), and a single byte as operation status (*OpStat*).

5.1. Latency

We define the *overall latency* as time elapsed from the moment an application on the DCD submitted a request to the moment the application receives a corresponding response from the KSD. The *overall latency* consists of two parts (a) *communication latency* — the time spent on transferring all four messages over BLE, and (b) *computation latency* — the time spent on all required computations, such as encryption, decryption. We ignore *computation latency* from further analysis, because the results of our benchmarking experiments showed that *computation latency* is in order of several magnitudes smaller than the *communication latency*.

There are several fundamental factors in BLE that impact *communication latency*. First, maximum transmission unit size at the ATT layer (referred to as MTU_ATT in BLE specification [13]) limits the number of bytes one can fit into a single packet. By sniffing the BLE connection setup, we identified that CC2540 and Android OS support only 23 bytes in each packet, while iOS allows 132 bytes. Second, BLE defines a connection interval (CInterval), i.e., a time period during which only a single packet is allowed. Our experiments with iOS and Android OS revealed that the default values for CInterval are 30 ms and 48.5 ms respectively. Finally, BLE defines a connection interval latency (CLatency), which defines the number of allowed connection intervals without a message. Note that, to maintain connection, BLE stack sends so called *empty-PDU*s in unused connection intervals, CLatency allows devices to skip sending *empty-PDU* without dropping the connection. Skipping several intervals allows the wearable device to use power-efficient modes for longer periods of time, which overall reduces power consumption. By sniffing BLE communication, we identified that both iOS and Android OS use zero as the default value for CLatency, i.e., skipping connection intervals is not allowed by default. Results shown in Table 1 suggest that with default values of CInterval and the smallest MTU_ATT, the system introduces no more than a second of latency overhead.

5.2. Power consumption

Smartphones. Power consumption is a crucial property of systems that are meant to be deployed in a mobile environment. Draining too much power will make the proposal impractical. That is why we experimentally measured the power consumption overhead that the Sidekick system introduces. The results of our experiments revealed that retrieving a single 256-bit KEK consumes approximately 5.7 μAh on smartphones, or about 0.0004% of iPhone's battery capacity. This suggests that Sidekick system introduces negligible power consumption overhead for smartphones.

Table 1
Overall Latency for each four request/response message pairs for the default values for CInterval and CLatency.

Req/Resp,	Overall Latency, ms	
(Payload Length, bytes)	Android OS	iOS
Store (32/4)	873	540
Retrieve (4/32)	873	540
Update (32/0)	873	540
Delete (4/0)	776	480

Table 2
CR2032 battery life in days, depending on the maximum round-trip latency for any request and for different numbers of requests per day.

	Maximum Latency, ms			
Requests per Day	540	1000	1500	2000
1	14	217	443	652
10	14	217	442	651
100	14	215	434	633
1000	14	197	366	496

Wearables. CC2540 SoC is based on the 8051 CPU, which provides great flexibility in power consumption through four power modes: Active Mode, and Modes 1–3. In theory, CC2540 can run for 9 h in Active Mode and up to 30 years in Mode 3 on a single CR2032 battery [25]. Of course, in practice the battery lifespan depends on specifics of the application.

The results of our experiments revealed that a single KEK retrieval request consumes approximately $0.12\mu Ah$ of battery capacity on CC2540, which corresponds approximately to 0.00005% of battery capacity. That is, a single CR2032 battery allows the KSD to receive and process about 2 million requests from the DCD. To assess battery lifespan, we need to define a daily workload, i.e., the number of requests sent per day to the KSD. For example, if we consider that the KSD stores a single KEK for the entire smartphone, then we can set the expected daily workload to approximately 100 requests a day [34]. In other use cases, the expected workload can differ. For example, let us consider a case when only a banking app stores KEKs on KSD. For a banking app we can safely assume a user does not open it on every smartphone unlock, which reduces the expected daily workload. On the other hand, if multiple applications use Sidekick to store KEKs, then we would expect higher workload. To cover different scenarios, we defined four workloads with 1, 10, 100, and 1000 requests a day.

Another parameter that impacts power consumption is CInterval. If increasing overall latency is acceptable, one can increase the value of that parameter in order to make the wearable more energy efficient. To assess this possibility, we explored the following four values {0, 15, 32, 48} for CInterval parameter, which correspond to the following set of latencies 540, 1000, 1500, 2000 ms for the KEK fetching request.

The results shown in Table 2 suggest that the default parameters for the BLE channel (CInterval = 30 ms and CLatency = 0) for the CC2540-based KSD allow it to run for two weeks only. If, however, we increase maximum allowed latency up to 2000 ms then with the daily workload of 100 requests a day or less, the battery would last for more than 600 days.

5.3. *Session key renewal*

Security of the session key is crucial for overall security of the Sidekick system. Unfortunately, our experiments revealed that establishing a single 128-bit session key consumes 0.044% (or 0.1 mAh) of battery capacity on the CC2540 SoC. That is, a single CR2032 battery allows the establishment of 2272 session keys with ECDH/P128 at most. That is why we have to factor in the energy budget for session key renewals in our battery lifespan assessment.

For demonstration purposes let us consider the following example:[3] 100 requests a day are expected on average and two seconds overall latency is acceptable. We can see from Table 2 that the battery would last for 633 days at most with such requirements. If we agree that having a year of battery life for the wearable device is acceptable, then it leaves 42% of the battery capacity available for key renewal. Hence, one can see that one can establish session keys 973 times in 365 days, or slightly more than twice a day.

6. Related work

There are two approaches to improve data-at-rest security. First, one can increase the cost of the brute-force attack, by making key derivation process harder for an attacker. In fact, PBKDF2 has been replaced with *scrypt* in Android 4.4 for that exact reason. Second, defenders can persuade users to use passwords that are substantially harder to guess.

[3] This example closely matches the requirements of current data protection systems in iOS and Android OS.

Novel KDFs have been proposed, such that it substantially increases the cost of each step for the attacker. For example, Boyen [35] developed a halting key derivation function, which forces an attacker to do substantial amounts of additional computations. Other proposals (e.g., [10]) have suggested to increase the number of KDF iterations. This, however, is a never ending arm-race. In contrast, we proposed a KDF agnostic system, that is, one can use Sidekick to store actual KEKs and does not need to rely on a secure KDF.

Others focused on proposing and evaluating novel authentication methods on smartphones (e.g., [11]). The main objective of such proposals is to introduce an alternative, usable, yet secure way to unlock smartphones. The evaluation results suggest that the new proposals are usable and resilient against specific attacks, such as shoulder-surfing attacks. Unfortunately, these proposals still create authentication secrets comparable in entropy with conventional 4-digit PINs. In comparison, we present system that breaks that dependency on a strong authentication secret.

Finally, researchers proposed protection techniques for the data-at-rest on mobile devices. For instance, DOrazio et al. [36] proposed an approach to conceal unprotected data in iPhones. Their solution, however, requires substantial expertise from end-users in order to set up the concealment.

To summarize, all currently deployed systems for data-at-rest protection either unjustifiably rely on users to choose secure passwords or require substantial expertise. In addition, while certain threats, e.g., shoulder surfing, have been addressed by the research community, attempts to make a brute-force attack on passwords impractical have failed. To the best of our knowledge, we are the first to use wearable devices for a practical and effective solution of mitigating an attacker who steals a smartphone and attempts to decrypt confidential data.

7. Discussion and future work

Overall, the evaluation results suggest that our proposal is practical in both terms: added latency and power consumption. Low power consumption makes our proposal attractive for organizations since it requires minimal maintenance efforts, i.e., frequency of battery replacement. Considering that the mobile OS does not need to be modified and that our proposal is cross-platform immediate deployment is possible. In fact, the FusionPipe[4] company has adopted our proposal for data encryption in their Mobile Device Management solution.

Although we focused on the technical side of the system design, one can argue that a usable wearable device is required for our system to be adopted. We, however, envision the Sidekick system to be a part of wearable devices that smartphone owners already use in their daily life (e.g., fitness trackers or smartwatches). Unnoticeable memory footprint (less than 4 Kb/20 Kb and RAM/ROM) makes incorporation of Sidekick's functionality into existing wearables a trivial exercise. Such wearable devices, however, must be designed with security in mind, in order to keep KEK's storage properly protected (e.g., see [37] for boot chain exploit).

Finally, it is possible to use multiple KSDs to protect a single DEK. Use of multiple KSDs can increase usability and security. For example, consider a setup where each of **N** used KSDs allows decrypting DEKs on DCD. This would not only allow the end-user to recover from loss of one of the KSD easier, but also will give him a freedom of choice for KSD form-factor (e.g., a fitness tracker or RFID card).

To increase security, one can set up a policy that requires more than one KSD to access DEK. For example, one might want an employee to have his RFID card and his smartwatch on him in order to access corporate confidential data on a smartphone. Yet another scenario is when access to confidential data needs to be controlled by more than two persons, i.e., multiple persons authorization. In that case, KSD of each of the person is used for storing KEKs during encryption process, and all of them must be present within a certain proximity to be able to recover the DEK.

For the aforementioned scenarios we can employ the *k out of n* scheme with Shamir's Secret Sharing [38] which uses *n* KSDs, but requires only *k* of them to reconstruct KEK. Such deployment allows some flexibility in who can participate in the multiple persons authorization process. Using more than one device makes the job for the attacker significantly harder. The attacker would have to target not one but many devices for theft, and not one but many smartphone users in the case of multiple persons authorization.

8. Conclusion

In this work we presented the design and evaluation of the Sidekick system — a system that uses wearable devices for encryption key management. The proposed system is effective at mitigating the adversary who steals a smartphone and tries to decrypt confidential data-at-rest, since it substantially increases the key encryption key (KEK) search space. This renders the brute-force attack on KEK impractical.

In addition, the proposed system is practical. That is, it can be deployed across all mobile platforms right away, it works on new and old devices, it introduces unnoticeable power consumption overhead, and the introduced latency can be up to a few seconds, depending on power consumption requirements.

We evaluated the Sidekick system on iOS and Android platforms, and implemented the Key Storing Device (KSD) on a SoC common for released wearable devices (CC2540 [33]). The results of the evaluation revealed that the default parameters

4 http://www.fusionpipe.com/.

for BLE connection are not practical. To overcome this limitation, we proposed an approach to select the values of these parameters, based on required latency and battery lifespan. In an example case we demonstrated how CC2540 SoC can be configured to last for a year on a single CR2032 coin cell battery, while providing ability to refresh session key twice a day and keeping latency under two seconds.

References

[1] I. Muslukhov, Y. Boshmaf, C. Kuo, J. Lester, K. Beznosov, Understanding users' requirements for data protection in smartphones, in: Workshop on Secure Data Management on Smartphones and Mobiles, 2012.
[2] I. Muslukhov, Y. Boshmaf, C. Kuo, J. Lester, K. Beznosov, Know your enemy: the risk of unauthorized access in smartphones by insiders, in: Proceedings of the 15th International Conference on Human–computer Interaction with Mobile Devices and Services, MobileHCI'13, ACM, New York, NY, USA, 2013, pp. 271–280. http://dx.doi.org/10.1145/2493190.2493223.
[3] 2014 cost of data breach study, http://www-935.ibm.com/services/us/en/it-services/security-services/cost-of-data-breach/, 2015 (last accessed 22.04.15).
[4] Phone theft in america, https://www.lookout.com/resources/reports/phone-theft-in-america, 2015 (last accessed 22.04.15).
[5] Announcement of new initiatives to combat smartphone and data theft, https://www.fcc.gov/document/announcement-new-initiatives-combat-smartphone-and-data-theft, 2015 (last accessed 12.05.15).
[6] Smart phone thefts rose to 3.1 million last year, consumer reports finds, http://www.consumerreports.org/cro/news/2014/04/smart-phone-thefts-rose-to-3-1-million-last-year/index.htm, 2015 (last accessed 22.04.15).
[7] I. Cherapau, I. Muslukhov, N. Asanka, K. Beznosov, On the impact of touch id on iphone passcodes, in: Proceedings of the Symposium on Usable Privacy and Security, SOUPS'15, 2015, p. 20.
[8] R. Raguram, A.M. White, D. Goswami, F. Monrose, J.-M. Frahm, iSpy: automatic reconstruction of typed input from compromising reflections, in: Proceedings of the 18th ACM conference on Computer and communications security, CCS'11, ACM, New York, NY, USA, 2011, pp. 527–536. http://dx.doi.org/10.1145/2046707.2046769.
[9] Apple, iOS Security, 8.1 and up, http://www.apple.com/business/docs/iOS_Security_Guide.pdf, 2014 (accessed 26.04.2015).
[10] A. Skillen, M. Mannan, On implementing deniable storage encryption for mobile devices, in: Proceedings of the 20th Annual Network and Distributed System Security Symposium, NDSS Symposium'13, San Diego, CA, USA, 2013.
[11] A. De Luca, A. Hang, F. Brudy, C. Lindner, H. Hussmann, Touch me once and I know it's you!: implicit authentication based on touch screen patterns, in: Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems, CHI'12, ACM, New York, NY, USA, 2012, pp. 987–996. http://dx.doi.org/10.1145/2208516.2208544.
[12] Bluetooth Smart Technology: Powering the Internet of Things, http://www.bluetooth.com/Pages/Bluetooth-Smart.aspx, 2013 (accessed 15.02.14).
[13] Specification Of The Bluetooth System 4.1, https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=229737, 2013 (accessed 08.02.13).
[14] H.-Y. Chiang, S. Chiasson, Improving user authentication on mobile devices: A touchscreen graphical password, in: Proceedings of the 15th International Conference on Human–computer Interaction with Mobile Devices and Services, MobileHCI'13, ACM, New York, NY, USA, 2013, pp. 251–260. http://dx.doi.org/10.1145/2493190.2493213.
[15] Q. Do, B. Martini, K.-K. R. Choo, A forensically sound adversary model for mobile devices, PLoS ONE 10 (9) (2015) 1–15. http://dx.doi.org/10.1371/journal.pone.0138449.
[16] Elcomsoft iOS Forensic Toolkit, http://www.elcomsoft.com/eift.html, 2012 (accessed 15.02.13).
[17] J. Zdziarski, Identifying back doors, attack points, and surveillance mechanisms in iOS devices, Digit. Invest. 11 (1) (2014) 3–19.
[18] Bluetooth smart: the good, the bad, the ugly and the fix, https://www.blackhat.com/us-13/briefings.html#Ryan, 2013 (accessed 26.02.14).
[19] T. Rosa, Bypassing passkey authentication in bluetooth low energy, Cryptology ePrint Archive, Report 2013/309, 2013. http://eprint.iacr.org/.
[20] M. Ryan, Bluetooth: With low energy comes low security, in: Presented as part of the 7th USENIX Workshop on Offensive Technologies, USENIX, Berkeley, CA, 2013. https://www.usenix.org/conference/woot13/workshop-program/presentation/Ryan.
[21] M. Naveed, X. Zhou, S. Demetriou, X. Wang, C. Gunter, Inside job: Understanding and mitigating the threat of external device mis-bonding on android, in: Proceedings of the 21th Annual Network and Distributed System Security Symposium, NDSS Symposium'14, San Diego, CA, USA, 2014.
[22] TrueCrypt - Free Open-Source On-The-Fly Disk Encryption Software for Windows 7/Vista/XP, Mac OS X and Linux, http://www.truecrypt.org/, 2013 (accessed 15.02.13) Version 7.1a.
[23] BitLocker Drive Encryption. http://windows.microsoft.com/en-CA/windows7/products/features/bitlocker, 2014 (accessed 26.02.14).
[24] Bluetooth SIG Analyst Digest Q4 2012. https://www.bluetooth.org/library/userfiles/file/Bluetooth_Analyst, 2012.
[25] C. Gomez, J. Oller, J. Paradells, Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology, Sensors 12 (9) (2012) 11734–11753. http://dx.doi.org/10.3390/s120911734.
[26] Nike+. https://secure-nikeplus.nike.com/plus/, 2014 (accessed 26.02.14).
[27] Bluetooth Smart CC2541 SensorTag. http://www.ti.com/ww/en/wireless_connectivity/sensortag/index.shtml?DCMP=sensortag&HQS=sensortag-bn, 2014.
[28] D. Whiting, N. Ferguson, R. Housley, Counter with CBC-MAC (CCM). http://tools.ietf.org/html/rfc3610, 2003.
[29] J. Jonsson, On the security of ctr+ cbc-mac, in: selected Areas in Cryptography, Springer, 2003, pp. 76–93.
[30] N. Ferguson, B. Schneier, T. Kohno, Cryptography Engineering: Design Principles and Practical Applications, John Wiley & Sons, 2011.
[31] N. Saxena, J.-E. Ekberg, K. Kostiainen, N. Asokan, Secure device pairing based on a visual channel (short paper), in: Proceedings of the 2006 IEEE Symposium on Security and Privacy, SP'06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 306–313. http://dx.doi.org/10.1109/SP.2006.35.
[32] W. Diffie, M. Hellman, New directions in cryptography, IEEE Trans. Inf. Theory IT-22 (1976) 644–654. http://citeseer.nj.nec.com/diffie76new.html.
[33] Bluetooth Low Energy System on Chip CC2540. http://www.ti.com/product/CC2540, 2013 (accessed 15.02.13).
[34] V. Woollaston, How often do you check your phone? the average person does it 110 times a day (October 2013), http://www.dailymail.co.uk/sciencetech/article-2449632/.
[35] B. Xavier, Halting password puzzles – hard-to-break encryption from human-memorable keys, in: 16th USENIX Security Symposium—SECURITY 2007, Berkeley: The USENIX Association, 2007, pp. 119–134, available at http://www.cs.stanford.edu/~xb/security07/.
[36] C. DOrazio, A. Ariffin, K.K.R. Choo, ios anti-forensics: How can we securely conceal, delete and insert data?, in: System Sciences (HICSS), 2014 47th Hawaii International Conference on, 2014, pp. 4838–4847. http://dx.doi.org/10.1109/HICSS.2014.594.
[37] Q. Do, B. Martini, K.-K.R. Choo, Is the data on your wearable device secure? An android wear smartwatch case study, Software: Practice and Experience (2016). http://dx.doi.org/10.1002/spe.2414.
[38] A. Shamir, How to share a secret, Commun. ACM 22 (11) (1979) 612–613. http://dx.doi.org/10.1145/359168.359176.