HAT: Secure and Practical Key Establishment for Implantable Medical Devices

Sayon Duttagupta
imec-COSIC
KU Leuven, Belgium
sayon.duttagupta@esat.kuleuven.be

Eduard Marin
Telefónica Research, Spain
eduard.marinfabregas@telefonica.com

Dave Singelée
imec-COSIC
KU Leuven, Belgium
dave.singelee@esat.kuleuven.be

Bart Preneel
imec-COSIC
KU Leuven, Belgium
bart.preneel@esat.kuleuven.be

ABSTRACT

During the last few years, Implantable Medical Devices (IMDs) have evolved considerably. IMD manufacturers are now starting to rely on standard wireless technologies for connectivity. Moreover, there is an evolution towards open systems where the IMD can be remotely monitored or reconfigured through personal commercial-off-the-shelf devices such as smartphones or tablets. Nevertheless, a major problem that still remains unsolved today is the secure establishment of cryptographic keys between the IMD and such personal devices. Researchers have already proposed various solutions, most notably by relying on an additional external device. Unfortunately, these proposed approaches are either insecure, difficult to realise in practice, or are unsuitable for the latest generation of IMDs. Motivated by this, we present HAT, a secure and practical solution to provide *fine-grained* and *dynamic access control* for the next generation of IMDs, while offering full control and transparency to the patient. The main idea behind HAT is to shift the access control responsibilities from the IMD to an external device under the user's control, such as a smartphone, acting as the IMD's Key Distribution Center. We show that HAT only introduces minimal energy and memory overhead and formally prove its security using Verifpal.

CCS CONCEPTS

• **Security and privacy** → **Cryptography**; **Key management**; **Authentication**; *Security protocols*; • **Computer systems organization** → *Embedded and cyber-physical systems.*

KEYWORDS

Implantable Medical Devices, Cryptography, Pairing, Key agreement protocol

ACM Reference Format:
Sayon Duttagupta, Eduard Marin, Dave Singelée, and Bart Preneel. 2023. HAT: Secure and Practical Key Establishment for Implantable Medical Devices. In *Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy (CODASPY '23), April 24–26, 2023, Charlotte, NC, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3577923.3583646

1 INTRODUCTION

Millions of people worldwide rely on Implantable Medical Devices (IMDs), such as pacemakers, insulin pumps or neurostimulators, to monitor and treat a broad range of chronic conditions. Over the past decade, most IMDs have started incorporating radio transceivers to allow external devices to wirelessly collect patient health information and/or reprogram the IMD's settings without requiring invasive surgery. In the first generation of wireless-enabled IMDs, the set of devices an IMD could communicate with was limited only to dedicated and specialised *device programmers* (e.g., [8]) and *base stations* (e.g., [29]) manufactured by the IMD vendor. Both device programmers and base stations had a built-in programming head that activates the IMD's wireless interface when it is placed above the implantation site for a few seconds. Once activated, the IMD and the programmer/base station communicate with each other over a long-range communication range (2-5 meters away). Previous work discovered that these first generations of IMDs with wireless capabilities typically rely on proprietary (i.e., non-standard) wireless communication protocols with *no* security mechanisms embedded [15, 23, 25–27]. Consequently, anyone who can get in close proximity of the patient could activate their IMD and send valid commands to it.

Unlike the first generations of such IMDs, the latest generation can communicate with a much broader set of devices using standard wireless technologies like Bluetooth Low Energy (BLE). The advantage of standard communication technologies over proprietary wireless communication protocols is that they are supported by most personal, commercial-off-the-shelf devices such as smartphones or tablets. *Therefore, the set of devices that can be connected to the latest generation IMDs is no longer limited to a few dedicated and specialised devices.* This translates to today's IMDs being able to interface with a much broader set of devices, enabling the realisation of new use cases towards more effective and personalised healthcare. However, this evolution towards open and standard communication systems also puts IMD security at stake. While the security of the latest generation IMDs has not yet been investigated in depth by the research community, *it is evident that the increased connectivity of IMDs enlarges the attack surface significantly and brings new security risks and threats to the IMD ecosystem.*

Cryptographic solutions are thus needed to secure the wireless communication channel between the IMD and personal devices. However, securing the data in transit first requires the establishment and management of the cryptographic keys used to bootstrap the secure communication channel. This is a challenging problem and solving it is the main goal of this paper.

Research gap. Several works have proposed key establishment solutions for IMDs (see Section 9 for more details). Unfortunately, past works found that existing security solutions are either insecure (e.g., [24, 34, 42]), undesirable to patients [10] or not technically or practically viable (e.g., [13]). Besides, current solutions are tailored for the first generation of wireless implants where the IMD typically interacts with a very small and fixed set of external devices. Hence they are unable to provide dynamic and fine-grained access control as required by novel personal healthcare use cases.

Motivation. During their lifetime, the latest generation of IMDs will need to establish keys with multiple personal devices with whom no prior trust relationship exists. For example, during the lifetime of the IMD, the patient may change smartphones a few times. When this happens, the old smartphone should be revoked and the new device should get access to the IMD. Similarly, imagine that the patient's usual doctor is not available for an appointment, and the patient needs to go to one of her colleagues. In that case, the patient's usual doctor could provide her colleague with the necessary permissions needed to access the device for a given time. A similar situation would occur when the patient would get ill on holiday, or when an emergency would take place. In these cases, another doctor would need temporary access to the IMD. It is worth noting that doctors will also most likely change their personal devices during the IMD's lifetime, requiring the revocation of the doctor's former personal device and granting access to the new device. Another scenario that could occur, is that the patient would change the hospital or doctor, for example, when moving out to a new city or country. If this would happen, one might want to revoke access to all devices of the former hospital where the patient was going and grant access to the doctor's personal device in the new hospital. All these examples show the need for dynamic and fine-grained access control. In this context, it is fundamental for patients or caregivers to have full control over which personal devices can be connected to an IMD at a given time and have the ability to revoke devices or temporarily allow a personal device access to the IMD .

Goals and challenges. To the best of our knowledge, we are the first to present a secure *and* practical solution for the latest generation of IMDs. Inspired by the principle of least privileges, the core idea of our solution is to keep the number of devices that can access the IMD as small as possible at all times while guaranteeing access to the IMD by legitimate users whenever needed. In designing our solution, we had to overcome two main challenges:

First, IMDs lack I/O interfaces, such as a keyboard or a display, and are not physically accessible once implanted. As such, IMDs are not capable of evaluating access control requests from personal devices. This becomes particularly challenging if the set of external devices that want to communicate with the IMD changes over time. Also because of these reasons, conventional device pairing solutions, for example as specified in the BLE standard, cannot be applied in the context of an IMD; they would result in the use of a fixed password or PIN (which is insecure) or would shift the pairing problem towards using an out-of-band channel (which is not further specified in the standards). Based on these observations, we propose to outsource the access control functionality to an external device operated by the user (or someone they trust). Although there are a few key establishment solutions proposed using external devices (see Sect. 9.1 for more details), these are either insecure or require buying and wearing additional hardware. In contrast, our approach solely relies on the use of an external device that the user already has to support key establishment.

Second, one also needs to consider that access to the IMD is strongly linked to a social context. A patient might allow the smartphone of a family member or an attending physician to connect to his IMD, but not the personal device of another (unknown) caregiver. As a result, centralised key management solutions, such as Kerberos, where one would use a central cloud-based Key Distribution Center (e.g., managed by the IMD vendor), are not suitable. Indeed, it would be impossible for a centrally managed KDC to know which external devices should be granted access to the IMD at a given time. Similarly to the first challenge, this becomes even more challenging when the set of personal devices that can communicate with the IMD changes over time, as new devices need to get paired with the IMD and some devices may need to be revoked.

CONTRIBUTIONS

In this paper, we present HAT, a secure and easy-to-deploy solution to establish and manage cryptographic keys for the latest generation IMDs. The main contributions of this paper are the following ones:

- Ability to delegate the IMD's access control and key management operations to a *security manager* – an external device under the patient's control (or somebody they trust) – that issues access tokens to securely establish session keys between personal devices and the IMD.
- Introduce a secure key-exchange protocol, relying on a hash-based access token, between personal devices and the IMD.
- Leveraging on the prior work in embedded security, we evaluate the security of our scheme and conclude that our solution can be implemented on any IMD with minimal energy and memory overhead.
- Finally, we formally prove the security of HAT using Verifpal [21].

2 THREAT MODEL AND ASSUMPTIONS

The focus of our work is on the establishment of cryptographic keys between the IMD and personal devices (e.g., the patient's smartphone or the doctor's tablet), by relying on an external device. The main attack we aim to prevent through our work is an impersonation attack where an unauthorised device succeeds in establishing a secure communication session with the IMD. To that end, adversaries will aim to impersonate a personal device, an external device, or both, towards the IMD. Our threat model considers both *passive adversaries* who can eavesdrop on the wireless channel as well as *active adversaries* who can additionally modify, inject or tamper with messages.

We assume that the external device is operated by a user – potentially by the patient himself – whom the patient trusts to take well-considered decisions on behalf of the patient, and has not been compromised. Similarly, personal devices are commercial off-the-shelf devices (e.g., a smartphone or a tablet) that are operated either by the patient or by the medical staff, or by anyone who wants to pair their device with the IMD. In both cases, these devices include a dedicated application – installed by the user from an app store – to perform the security operations required in our solution. We then rely on existing solutions to enable the user to verify the correct application is being installed, that other applications running on the external device do not have access to the cryptographic material used in the application, and that an external adversary cannot remotely control the application.

Moreover, we assume that (i) the IMD manufacturer has a secure process in place to generate and install pre-shared keys in the IMDs, (ii) there is a secure shipping process in place between the IMD manufacturer and the hospitals that are using the IMDs, (iii) any IMD that does get implanted in a patient has not been compromised by the adversary in advance. Note that these are common assumptions widely used by previous solutions, as otherwise, it would be impossible to protect IMDs against impersonation attacks.

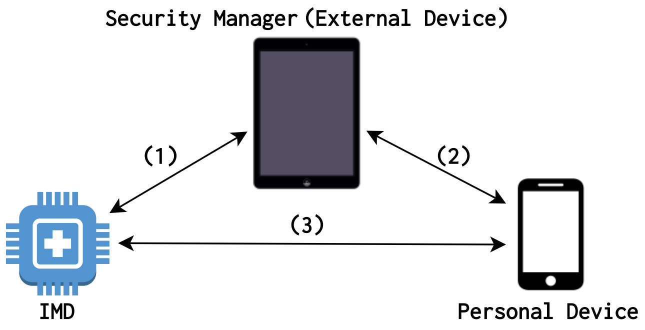

Figure 1: High-level HAT architecture. (1) Initial bootstrap sequence, only to be run once per security manager. (2) The security manager granting access-control rights to the personal device(s). (3) The personal device(s) agreeing on a session key with the IMD, without the security manager's presence.

3 HAT – CORE IDEA

The concept behind HAT is depicted in Fig 1. There are three main steps that need to be executed. The first step is to securely initialise an external device – i.e., the security manager – to manage the access requests to an IMD (see Sect. 4.1). It is important to note that this bootstrapping process only needs to be carried out on rare occasions. As a second step, the personal device must obtain an access token from the security manager. Finally, in the third step, the personal device uses the access token to establish a secure communication channel with the IMD (see Sect. 4.2). As in Kerberos, the security manager acts as the entity responsible for generating and distributing keys. However, one important difference worth noting is that the security manager does not need to be online at all times.

3.1 System Overview

Let us first briefly introduce the components and entities that are part of our solution.

Implantable Medical Devices (IMD): The IMD is a medical device which is inserted inside the patient's body through a surgical operation.

IMD Manufacturer: The IMD manufacturer or the vendor is the entity responsible for generating pre-shared initialisation keys of each IMD during fabrication, encoding these values in QR codes and securely shipping these to the hospitals together with the IMD itself. In the next sections, we discuss this process in more detail.

Patient: The person who has an implanted IMD.

Medical Staff: This comprises the hospital's medical staff, doctors and caregivers. They are the ones taking care of the patient, hence they might need access to the IMD's data or need to reconfigure it.

Personal Devices: A personal device can be any commercial off-the-shelf device, such as a smartphone or a tablet, which can be used to communicate with the IMD. These include all the devices used by the medical staff as well as the personal device(s) of the patient himself.

Security Manager: From a security point of view, one needs to manage the set of personal devices that are authorised to establish a secure communication session with the IMD. The external device responsible for managing this access control is the security manager. Any device can be used as the security manager once it has been securely initialised. For the rest of the paper's discussion, we will refer to the personal devices as solely the devices that want to communicate with the IMD after being granted access from the security manager, and the security manager being the policy enforcer. We will denote the security manager as the combination of the hardware of the external device (e.g., the smartphone of the user) and a dedicated application running on the device to perform the necessary operations required in our solution. In Sect. 4, we describe the functionalities and working of the security manager in more detail.

Security Manager operator: The person in control of the security manager (i.e., the one who is manually approving access control requests from personal devices that want to communicate to the IMD) is the security manager operator. Initially, the security manager operator can be the surgeon who implants the device, which then hands over the control to the patient (or anybody trusted by the patient). The security manager operator is also the person who will be in possession of the cryptographic material (QR codes) to securely initialise a security manager.

In the rest of the paper, we refer to the *security manager* as the external device solely responsible for evaluating pairing requests and granting access to users to pair with the IMD. Any other device which wants to pair with the IMD, we denote as a *personal device*.

3.2 Delegation of Access Control Functionality

As mentioned before, one of the root causes for the difficulty of realising key establishment is that the IMD is not capable of evaluating access control requests from personal devices due to the lack of sufficient context and I/O interfaces. We propose to securely outsource this functionality to an external device – *the security manager* – whose goal is to evaluate pairing requests and enforce security policies on behalf of the IMD. From a high-level point of view, the security manager will act as a Key Distribution Center (KDC) – locally controlled by the security manager operator – issuing access tokens for the IMD. Therefore, each time a new personal device needs to be paired to the IMD, it needs to communicate with the security manager first. For subsequent communication sessions with the IMD, the personal device can directly establish new session keys with the IMD, without interaction with the security manager. Therefore, only when a new personal device needs to be paired with the IMD, the security manager needs to be available. It is also important to stress that at each moment in time, for a specific IMD there is only one security manager in place; which can be replaced when needed.

One could argue that relying on a single device to control the key management operations and access to the IMD would introduce a single point of failure. However, we argue that there is nothing wrong with a single point of failure, as long as it fails rarely, and the consequences of a failure/breach can be easily mitigated. This is the case in our solution, as a user can decide at any time to assign the role of security manager to a new external device, and as such, automatically revoke the former security manager. Secondly, one can partially mitigate the availability risk by relying on cloud technology. We discuss both aspects later in the paper.

4 SECURITY MANAGER

In this section, we will zoom in on three important security aspects: (i) how to initialise a new security manager, (ii) how to use it to pair a new personal device to the IMD, and (iii) how to revoke a security manager and replace it by a new one.

4.1 Bootstrapping the Security Manager

When a personal device wants to establish a shared session key with the IMD, it has to present a valid cryptographic token issued by the security manager. To verify the validity of this token, the IMD should have established a trust relationship with the security manager prior to this. Otherwise, it will be impossible for the IMD to distinguish valid tokens from forged ones. This trust relationship will be established during the bootstrap process of the security manager, which can take place at any time once the IMD has been implanted in the patient. Inspired by the pairing method proposed by Stajano and Anderson in their seminal work [41], in HAT the first device that successfully carries out the secure bootstrapping process becomes the security manager. Note that this bootstrap process only happens occasionally.

During the fabrication of the IMD, the manufacturer will generate a set of unique, IMD-specific cryptographic keys. In particular: (i) a device-specific 256-bit master secret key ms, (ii) a 128-bit random private key k and (iii) *multiple* initialisation secret key-pairs (s_i, s_i^*). The latter are derived from the master key, a randomly-generated salt and the index i, using a Hash-based Key Derivation Function (HKDF). In particular, this is done as follows:

$$s_i = HKDF(\ ms_1 \mid\mid salt \mid\mid i\ ) \tag{1}$$

$$s_i^* = HKDF(\ ms_2 \mid\mid salt \mid\mid i\ ) \tag{2}$$

Where, $ms = ms_1 \mid\mid ms_2$. Here, we take the first 128 bits of ms to compute the keys s_i and the next 128 bits for the keys s_i^*. We use s_i as the initialisation key to our protocol and also to bootstrap the IMD with the security manager and generate the hash-chain. The other key of the key pair, s_i^*, is used as the key to the MAC algorithm for authenticating the messages generated by the security manager for the IMD, thus guaranteeing *key-indistinguishably*.

The pairs (s_i, s_i^*) are stored on the IMD, together with the corresponding IMD's public key $k\mathcal{P}$ (more details in Sect. 5). Meanwhile, (s_i, s_i^*) and the IMD's public key are encoded in multiple, sequentially numbered (using the index i) QR codes that are put on the inside of the package used to ship the IMD to a hospital[*]. To avoid anybody could read out a key during transportation, the QR codes are sealed by a layer that can be scratched or peeled away in the hospital after the package has been unpacked. After the IMD has been implanted, the patient must store the remaining QR codes securely to be able to revoke or bootstrap a new security manager at any time in the future, if needed.

To bootstrap the security manager, the security manager operator just needs to read out the next fresh QR code on the IMD's package to learn the cryptographic credentials. Let us assume that the next QR code available is i. Using the initialisation secret s_i, the security manager will perform an initial bootstrap protocol with the IMD, shown in Fig.2, to establish a new secret key s_i', and to create access tokens. The details of the initial bootstrap protocol will be discussed in the next section. Note that for security reasons, each key-pair (s_i, s_i^*) is only used once during the initial bootstrap protocol and to securely transport security-critical messages (e.g. to revoke a personal device). Once the bootstrap protocol has finished, the key-pair (s_i, s_i^*) cannot be used to initialise another security manager. After bootstrapping the security manager, we would establish a secure channel to start the security manager granting access tokens to other devices. We narrate this process in detail in Sect. 5.1.

As mentioned above, the credentials (initialisation keys) in all the QR codes that have not yet been used, are securely stored (digitally or physically). For example, this can be done by storing all the QR codes in a safe, or by storing the digital information online in a secure vault. It is important that no unauthorised party can access these credentials and their availability is guaranteed at all times (i.e., backups should be available). The only difference between the security manager and any other device, from a security point of view, is that the former had access to the QR codes and/or its credentials, and the latter does not.

[*]Both keys s and s^* are embedded inside a single QR-code and to bootstrap, a security manager the user only has to scan a QR-code once; we believe this is a relatively easy task that most users can perform.

4.2 Using the Security Manager

After bootstrapping it, the security manager will be ready to evaluate pairing requests sent by personal devices. Next, we illustrate the process through which a personal device pairs to the IMD with the help of the security manager.

(1) The first step is for the personal device to get temporarily paired with the security manager. Since both devices are typically mobile personal devices, one could use any pairing techniques that offer good security and usability properties. For example, one could use, for example, Manual Authentication (MANA) protocols in combination with an out-of-band channel (e.g., a telephone call between the security manager operator and user of the personal device) [28, 37]. The result of this pairing procedure is that a secure channel is established between the security manager and the personal device (e.g, a TLS channel in case both devices are remotely connected over the Internet).
(2) The personal device then sends a pairing request to the security manager over the secure channel.
(3) The security manager evaluates the pairing request and decides whether access to the IMD should be granted. This evaluation could rely on the consent of the security manager operator – taking into account social considerations, and/or be based on contextual properties (e.g, location and time of the request).
(4) If the pairing request is granted, the security manager creates a unique cryptographic token and sends this back to the personal device over the secure channel. At this point, the security manager's job ends. From this moment onward, if access to the IMD is needed, the personal device presents the cryptographic token to the IMD, which will evaluate the validity of the token. If the token is valid, the personal device can then carry out the cryptographic key establishment protocol we propose in Sect. 5.3 to establish a secret session key with the IMD.

It should be stressed that the procedure described above has to be executed only once for every new personal device that needs to be paired to the IMD. Once a session key is established, one can rely on conventional protocols to regularly refresh this key.

The overview above shows that the problem of the IMD having to evaluate access control requests is shifted to the verification of a cryptographic token. We argue that the latter is significantly less complex for the IMD, and allows the enforcement of more complex and dynamic access control policies by the security manager on behalf of the IMD. In the next sections, we describe the building blocks of our security solution in more detail.

4.3 Revoking the Security Manager

To guarantee availability, it should always be possible to revoke the current security manager and replace it with a new device, e.g., when it gets stolen or compromised. If this happens, the security manager operator will scan any of the remaining unused QR codes and will use the credentials encoded in them to run the bootstrapping protocol. Let us assume the next available QR code to be the number $i + 1$. The IMD will then check the freshness of the new initialisation secret key s_{i+1} (i.e., that it has not been used before). As a result, the system will be reset and the IMD will initialise fresh, independent keys s_{i+1} and s^*_{i+1} that are also known by the new security manager. Once the IMD and the new security manager have established this key-pair, the former security manager is revoked by default. The IMD will erase the old keys it shared with the revoked security manager from its memory so that the latter can no longer use it to communicate with the IMD and/or issue access tokens. By deleting the key s_i from its memory, all the tokens computed with this seed are also rendered useless. The IMD will also delete all the existing session keys it had stored in its memory, i.e. the access to all personal devices is revoked. This makes revocation in HAT effortless, one simply needs to execute the bootstrapping protocol with new keys obtained from any of the remaining, unused QR codes.

5 HAT: IMD KEY ESTABLISHMENT USING HASH-CHAIN TOKENS

In the two previous sections, we introduced the basic concepts of our security solution and described the high-level communication flow. In this section, we will now zoom in on the cryptographic protocols that need to be executed by the security manager and personal devices, respectively, to communicate securely with the IMD. In Sect. 5.1 we initialise the security manager with the IMD to be its *cryptographic token manager* and set up the initial seed and hash tokens to be distributed to the personal devices. In Sect. 5.2, we describe how the security manager evaluates pairing requests and performs delegation and revocation of session keys. In Sect. 5.3 we describe a secure and practical cryptographic key establishment protocol on how to agree on a shared session key to secure the wireless communication link, between the personal device and the IMD. This protocol combines elliptic-curve-based static-ephemeral Diffie-Hellman with hash-chain-based access tokens to provide implicit authentication of the ephemeral Diffie-Hellman key of the personal device.

Setup. Before explaining the working of our protocols in detail, let us first introduce the cryptographic primitives we use. For efficiency reasons, we rely on elliptic-curve cryptography to establish a session key. The domain parameters of the curve $\#E(\mathbb{F}_p)$ include: a prime finite field $\mathbb{F}_p$, the curve coefficients $a, b \in \mathbb{F}_p$, a base point $\mathcal{P} \in E(\mathbb{F}_p)$ generating a cyclic subgroup of large order, the order n of the sub-group and the co-factor $h = E(\mathbb{F}_p)/n$. These domain parameters are publicly known by all devices in the system, including the IMD. Another public parameter in the system is the pre-defined public initialisation sequence *init*. This is an arbitrary bit sequence that triggers the IMD to start the bootstrapping process (Sect. 4.1). During fabrication of the IMD, a unique master secret key *ms*, a set of initialisation key-pairs (s_i, s^*_i), and a random private key k are generated for the elliptic curve cryptosystem. The keys (s_i, s^*_i) are stored on the IMD, together with the corresponding IMD's public key $k\mathcal{P}$. The IMD also stores a counter c, which is initialised to zero, and is used later in the protocol for synchronisation. Finally, each secret key s_i, along with the MAC-key s^*_i and the public key $k\mathcal{P}$ are embedded in QR codes that are shipped together with the IMD. We will now discuss the different stages of the HAT protocol.

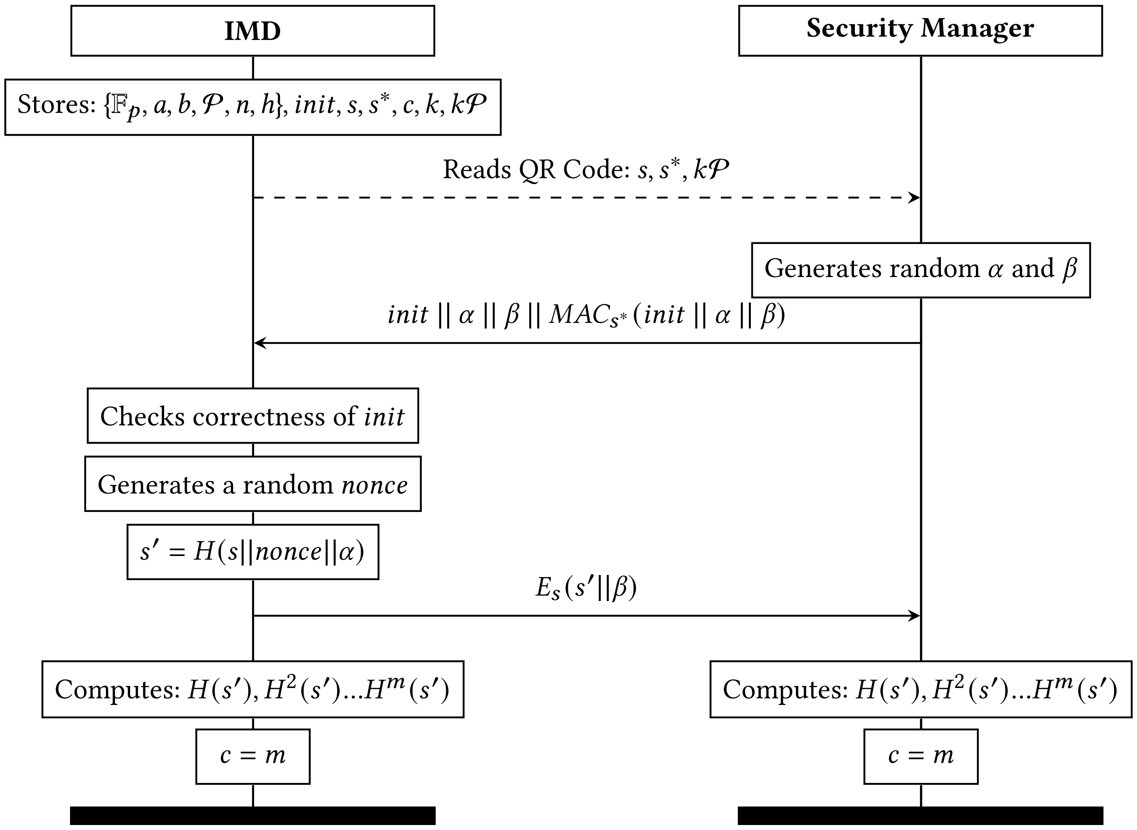
IMD
Security Manager
Stores: $\{\mathbb{F}_p, a, b, \mathcal{P}, n, h\}, init, s, s^*, c, k, k\mathcal{P}$
Reads QR Code: $s, s^*, k\mathcal{P}$
Generates random α and β
$init \,||\, \alpha \,||\, \beta \,||\, MAC_{s^*}(init \,||\, \alpha \,||\, \beta)$
Checks correctness of $init$
Generates a random $nonce$
$s' = H(s||nonce||\alpha)$
$E_s(s'||\beta)$
Computes: $H(s'), H^2(s')...H^m(s')$
$c = m$
Computes: $H(s'), H^2(s')...H^m(s')$
$c = m$

Figure 2: Initial bootstrap sequence between the IMD and the Security Manager.

5.1 Secure channel between the IMD and the Security Manager

The first step in our system is for a device acting as a security manager to pair with the IMD using the bootstrapping protocol we propose (see Fig. 2). To initialise the security manager, the security manager operator will first read any arbitrary unused QR code to obtain the secrets s and s^*, and the IMD's public key. Note that for ease of presentation, in the following we focus only on one specific instance of the bootstrapping phase; due to this, we leave out the index i in our notation. The security manager then generates two random values, α and β and sends them along with a pre-defined initialisation sequence $init$, and the MAC of the three values keyed using the initialisation secret s^*. The IMD will verify the MAC of the incoming message with the key s^* and check the correctness of $init$. If the check succeeds, the IMD will generate a fresh random $nonce$, to generate the seed and add randomness to the hash-chain. The IMD will then compute this seed s' as $s' = H(s||nonce||\alpha)$ by appending the values $nonce$ and α to the secret key s and applying a cryptographic hash function H. The secret s' is sent along with β from the IMD to the device, encrypted using key s. The security manager sends two random values to the IMD, α to add more randomness to the final seed to protect against bad randomness in the IMD, and β to act as a message acknowledgement and to offer mutual entity authentication. After receiving the message from the IMD, the security manager will decrypt the message and confirm the value β. The key s' is then used as the initial seed for the hash-chain access tokens. Both devices will compute the hash-chain by repeatedly hashing the initial seed s' - m times, using the cryptographic hash function H. The devices will then securely store all the m hashes. The parameter m should be carefully chosen during design time in such a way that one realistically cannot run out of tokens during the IMD's lifetime. Both the IMD and the newly initialised security manager will also update their synchronisation counter c with the current value of the topmost hash stored, i.e., $c = m$. Finally, both the security manager and IMD store the authentication key s^* in their memory, and use it to secure security management data.

5.2 Key management operations performed by the Security Manager

Granting access tokens. The security manager is responsible for authenticating the users of the personal devices (patients and/or medical staff) and granting access-tokens to their personal devices. After authenticating, the security manager will send the required credentials – the IMD's public key $k\mathcal{P}$, the hash counter c' and its corresponding hash-chain access token $H^{c'}(s')$ via a secure channel.

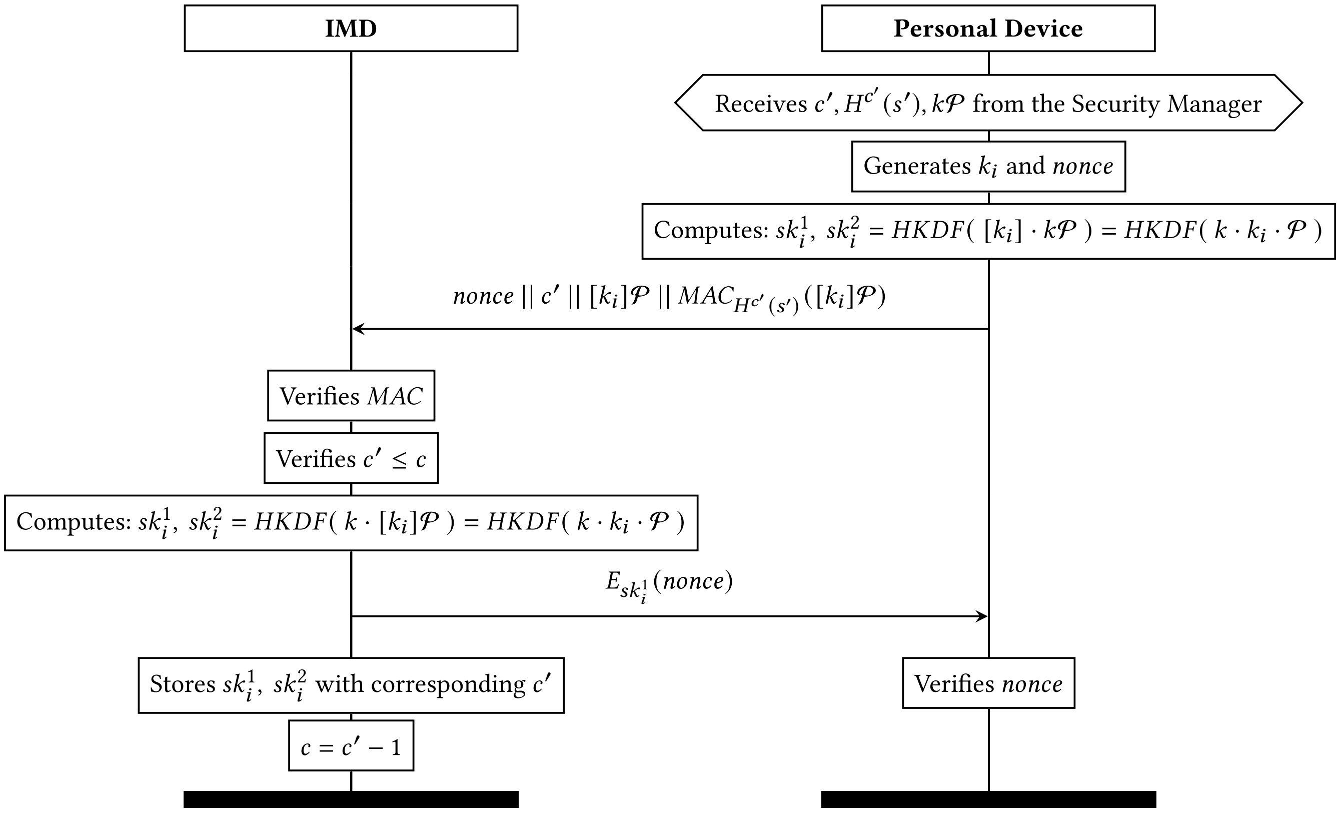
IMD
Personal Device
Receives $c', H^{c'}(s'), k\mathcal{P}$ from the Security Manager
Generates k_i and $nonce$
Computes: $sk_i^1,\ sk_i^2 = HKDF(\ [k_i] \cdot k\mathcal{P}\ ) = HKDF(\ k \cdot k_i \cdot \mathcal{P}\ )$
$nonce \,||\, c' \,||\, [k_i]\mathcal{P} \,||\, MAC_{H^{c'}(s')}([k_i]\mathcal{P})$
Verifies MAC
Verifies $c' \leq c$
Computes: $sk_i^1,\ sk_i^2 = HKDF(\ k \cdot [k_i]\mathcal{P}\ ) = HKDF(\ k \cdot k_i \cdot \mathcal{P}\ )$
$E_{sk_i^1}(nonce)$
Stores $sk_i^1,\ sk_i^2$ with corresponding c'
$c = c' - 1$
Verifies $nonce$

Figure 3: Key-exchange between the IMD and a personal device.

After issuing the hash-chain access token, the security manager will decrement the current value of its hash synchronisation counter c'.

Key delegation. In HAT, it is possible to further delegate access control to other devices, by providing them with one or more fresh hash-chain access tokens. The security manager can provide n hash values to another personal device using a secure channel, along with all the public parameters, such that the new device can act as a "delegate of the security manager" for a while (i.e., until all the n tokens are used). The security manager then decrements its hash synchronisation counter c by n. This delegate will now have n access tokens to distribute to other devices that want to communicate with the IMD. These devices will proceed to perform the same key agreement protocol as explained before and denoted in Fig. 3.

Session key revocation. If the security manager wants to revoke the session key of a device i to which it has previously granted access, it sends a revocation message to the IMD with the issued hash counter value. This message is authenticated using the authentication key s^*. The IMD then deletes the corresponding session key sk_i shared between the IMD and the device i. The revoked device i cannot re-establish a session key with the IMD without contacting the security manager. It can no longer use its old hash-chain token, as this hash has already been used (i.e., it will no longer be accepted by the IMD). Thus, it needs to ask for a new fresh hash-chain token from the security manager again. Similar to the public-key scheme, if the security manager is physically unable to transmit the revocation message to the IMD, it could be relayed through another personal device.

5.3 Key agreement between the IMD and the Personal Device

Every personal device needs to establish a session key with the IMD to start communicating with it[†]. The personal device will perform a key-exchange with the IMD as shown in Fig. 3 to agree on a session key.

After receiving the required tokens from the security manager, the personal device can now start a key-exchange protocol with the IMD. First, it will randomly generate a secret key k_i and a nonce, *nonce*, and then, using the domain parameters of the curve will compute the point $[k_i]\mathcal{P}$ as its public key. It will create a MAC of its public key, using the hash-chain access token as the key: $MAC_{H^{c'}(s')}([k_i]\mathcal{P})$. Note that since the IMD also knows the hash value, it can check the authenticity of this MAC. Next, the personal device will compute the shared point $[k_i] \cdot k\mathcal{P} = k \cdot k_i \cdot \mathcal{P}$, will discard the y-coordinate, and take the x-coordinate and pass it through a Hash-based Key Derivation Function (HKDF) with a random public salt, to derive two keys sk_i^1, sk_i^2. We are deriving two

[†]If the personal device and the IMD already share a session key, it can easily be refreshed at regular intervals using standard cryptographic techniques.

keys from the computed point to guarantee *key-indistinguishably*, by using one key for authentication and key confirmation and the other one as the session key for actual communication. It will then send its public key, along with the MAC of the public key, the hash counter c' and the *nonce* to the IMD. The IMD will first check if the hash counter c' it received is lower than the current counter value c stored in the IMD. If it is a fresh value, then it verifies the MAC message using the c'^{th} hash in the chain.

If successful, the IMD will update the counter value to $c' - 1$, so that the value can never be reused, and then compute the shared point $k \cdot [k_i]\mathcal{P} = k \cdot k_i \cdot \mathcal{P}$, and also derive two keys from the x-coordinate of the computed point. Finally, the IMD will encrypt the *nonce* and send it back to the personal device, encrypted using one of the new shared key sk_i^1. The IMD will then verify the nonce, hence providing mutual key confirmation. The IMD will store the keys (sk_i^1, sk_i^2) with the corresponding hash-token value in its memory. After authentication and key confirmation, it will use the second key to communicate with the device. The lifetime of these session keys depends on the protocol execution. Session keys that are not updated for a long time might be susceptible to long-term leakage. However, frequent revoking of short-span session keys puts an additional load on the IMD and increases its overall energy consumption. As IMDs, in general, are energy-constrained devices and are implanted within a body for several years, for efficiency reasons we opt to use the session key for a long lifetime and update it only when required. If a leak would happen, and this is known to the patient, then the security manager can easily revoke that particular personal device.

6 EVALUATION

6.1 Security assessment

The security of our scheme is mostly based on the difficulty of the computational Diffie–Hellman problem [3] over elliptic-curves. It assumes that the hash function H is a pre-image resistant one-way function and that each QR code is securely stored by the user. As each IMD is fabricated with a different long-term key k and is located inside the body, retrieving the secret data from an IMD in use is impractical. We recommend the use of the elliptic-curve Curve25519, which uses a Montgomery curve defined over a 255-bit prime field and achieves a security level of 128-bit and to use AES-128 as the block cipher for the session key encryption. We also recommend that all random nonces and (session) keys in the protocol are at least 128 bits long.

We have formally evaluated our cryptographic protocols first using Verifpal [21], and then validated the results obtained using ProVerif [4]. Both are widely used formal verification tools within the research community. In particular, we analysed the security of HAT against multiple unbounded instances of both passive adversaries, who can eavesdrop on a channel to gather information, and active adversaries who can actively monitor, spoof, inject or tamper with messages sent between the devices. Our formal analysis results show that our protocols are secure against multiple forms of confidentiality, authentication and spoofing attacks. A more detailed overview of our formal analysis can be found in Sect 7.

6.2 Performance analysis on the IMD

The HAT scheme is particularly designed for a resource-constrained embedded device. To demonstrate this, we analyse the performance cost of realising HAT on an MSP430, a representative mid-range microcontroller which is the reference platform for IMD research [36]. We split the energy cost of HAT into computation and communication costs, and analyse its memory overhead as well.

Computation cost. In HAT, the main cryptographic operations performed by the IMD are: (1) symmetric-key encryption and MAC algorithm, (2) hash function, and (3) elliptic-curve (EC) point multiplication. Even though multiple hash values have to be computed, it is well known that the energy consumption of symmetric-key encryption/MAC algorithm and hash functions is several orders of magnitude lower than EC point multiplication. An example of performance results of symmetric-key primitives on MSP430 can be found in [6]. Therefore, we can safely ignore the cost of all symmetric-key and hash operations and solely focus on the energy cost of the *single EC point multiplication* that is needed in the Diffie-Hellman key establishment protocol. Previous work has already extensively shown that it is feasible to implement elliptic-curve cryptography (ECC) on the MSP430 family of microcontrollers. As a reference example, we take the work of Hinterwälder et al. [16], who implemented ECC using Curve25519 on the MSP430F2618, with a 16-bit RISC CPU, 116 KB FLASH, 8 KB SRAM and a clock frequency of 16 MHz. Their implementation shows that a single EC-point multiplication could be executed in 9,139,739 clock cycles on the MSP430 microcontroller, with a 128-bit security level. The average power consumption was 14.05 μW and the average energy consumption was 11.62 μJ. This is orders of magnitude lower than the energy stored in a simple battery (e.g. 2.5 KJ in a coin cell). Note that the results cited above are very similar to other related work [43, 44], which also report an implementation cost of roughly 9 million clock cycles for an EC multiplication on an MSP430. To make the scheme more efficient, we have initialised the public key of the IMD during fabrication, hence using one less point multiplication.

Communication cost. Besides the computation cost, one also needs to consider the energy cost of the communication required in HAT. To analyse this cost, we assume that the communication between the IMD and a personal device is based on BLE, which is mostly the case in practice. The MSP430 microcontroller relies on the CC2560 chip from Texas Instruments to realise BLE communication. Krug and O'Nils [22] measured the current for sending or receiving data over BLE on this chip, which is respectively 5.3 and 6.4 mA. Using these numbers, one can compute the energy cost, which is respectively 18 nJ/bit and 22 nJ/bit for sending and receiving a single bit. Assuming all keys and nonces to be 128-bit, a unique ID of 64 bits and a counter of 16 bits, the total communication cost of HAT is 12.51 J per key-exchange instance. Again, this energy cost can be neglected compared to the total energy stored in the IMD's battery.

Memory overhead. In addition to energy, the impact on memory is also relevant. The IMD has to store the entire hash-chain, with n values, in its memory. The value of n depends on the way the IMD is used (e.g., based on the patient's condition), and can range from several hundred to a few thousand. The total number of hashes is a system parameter that can be adjusted depending on the lifespan of the IMD and should be large enough so that the IMD can realistically never run out of fresh hash values. However, the impact on storage is minimal. Jakobsson [18] and Coppersmith and Jakobsson [9] introduced a novel optimisation technique to compute and store an entire hash-chain with a given constant seed and achieved an upper bound of the complexity of $O(log^2 n)$, where n is the length of the chain. By using this optimisation, and assuming a counter of 16 bits (i.e., $n = 65536$), the storage cost 608 bytes, which is negligible on an MSP430 microcontroller. If we output one hash value per day, then the chain would last for 180 years. Alternatively, one can choose to compute the hash-chain on the fly for every protocol execution instead of pre-computing the hash-chain and storing the values in its memory. As computing a hash-chain is a lightweight operation, we believe this can minimise the total memory utilisation and optimise it further.

7 FORMAL VERIFICATION OF THE PROTOCOL

This section provides a detailed explanation of our formal analysis of HAT. Verifpal [21] is a new software tool used to formally verify the security of cryptographic protocols which is heavily inspired by ProVerif [4] and has already been used to analyse the security of complex protocols such as Signal and the DP-3T decentralized pandemic-tracing protocol [21]. Verifpal relies on a symbolic model that does not check for computational soundness and assumes the cryptographic primitives and functions are perfectly secure. Apart from being a very intuitive language, another key property of Verifpal is that it does not allow the design of custom cryptographic primitives, thereby avoiding well-known mistakes by users. Based on the properties above, we have opted to use Verifpal to analyse the security of our protocols.

The first step to formally verify the security of the bootstrap and key agreement protocols was to model them using Verifpal. We have attached the formal verification code in an anonymous repository and also uploaded both protocols to Verifpal's repository - VerifHub[‡]. In our formal analysis, we consider both *passive adversaries* who can only observe the protocol as well as *active adversaries* who can additionally modify, inject or tamper with the exchanged messages. We made multiple small modifications to our protocol codes so that they could be tailored to Verifpal's model. For example, instead of using point multiplication over an elliptic-curve for a Diffie-Hellman key-exchange, which is not supported by formal verification tools, we had to use exponentiation over a finite cyclic group. This is not an issue, as the underlying discrete-log problem remains the same in both cases. Our Verifpal code does not include any counters since Verifpal is unable to check for inequality, greater/lesser than or increment/decrement of variables. Instead of generating the entire hashchain $H(H(H(....H(x))))$, we hashed five times and used the fifth value for creating the hash-based tokens and simulating the model. In further analyses, the fourth, then the third token, and so on, are used. As it is not possible to depict an out-of-band or a secure channel in Verifpal, we explicitly declared some constants (like s) as known, private values in both entities. In practice, however, these values will be retrieved by reading a QR code, as described earlier in the paper. We have also assumed the IMD's public key is pre-authenticated (again, it is retrieved by a QR code in practice), but not for personal devices.

We formally verified the protocols in the Dolev-Yao model and we mainly tested and analysed our models for secrecy, authentication, spoofing attacks, trace and equivalence properties. Trace properties are defined on each protocol run. The protocol satisfies such a property when it holds true for all traces. E.g., the fact that some states are unreachable is a trace property. Equivalence properties mean that the adversary cannot distinguish two processes. For instance, one of these processes can be the protocol under study, and the other one can be its specification. Then, the equivalence means that the protocol satisfies its specification. Therefore, equivalences can be used to model many subtle security properties. We observed that Verifpal requires 1200 and 910 milliseconds, respectively, to test the bootstrap and key-exchange models. Our model passed all the tests and queries challenged to the verifier in both the passive and active attacker modes and gives 100% confidentiality, authentication and freshness success for all the values and parameters. Therefore, given the correctness of the cryptographic primitives being used, our solution provides a secure key-exchange protocol, with no confidentiality losses and with mutual authentication between all parties. Given Verifpal's goal of mimicking and resembling attacks on "real-world protocols", these outcomes support our claim that the protocol is secure against most known types of confidentiality and authentication attacks, such as Man-in-the-Middle, replay attacks, key compromise impersonation[§], across multiple unbounded protocol session executions.

Due to the fact that Verifpal is based on a rather intuitive high-level language that facilitates the writing and validation of cryptographic protocols, we opted to focus mostly on Verifpal to formally evaluate the security of our solution. Nevertheless, to further verify the security properties of HAT, we have also implemented all our protocols using ProVerif. By doing this, we were able to observe that both Verifpal and ProVerif provide identical results, again confirming the security of our proposed solution.

Both the Verifpal and ProVerif codes are available in the following link: https://gitlab.esat.kuleuven.be/Sayon.Duttagupta/HAT

8 DISCUSSION

In this section, we will discuss the salient features of HAT, apart from its security benefits, which makes it a robust and practical solution for the real world.

8.1 Feasibility of HAT

The entire scheme of agreeing on shared session keys or distributing hash-chain access tokens can be performed by a user-friendly app running on the security manager and it does not involve any complex manual procedures. Thus, users do not need to know or understand the inner workings of our solution to be able to use it. Moreover, the scheme can be carried out with off-the-shelf commercial personal devices, such as smartphones or tablets, and does not require any additional devices or bulky controllers. Moreover,

[‡]https://verifhub.verifpal.com/e3a201bd3f9c8161deefc41a9afde7f4
https://verifhub.verifpal.com/584744e1f37007812c654373740afbf6

[§]This property is only achieved if the IMD's private key is not leaked during the IMD's lifetime.

HAT allows us to have fine-grained and dynamic access-control privileges, and we can effortlessly re-initialise new devices and switch security managers. We also allow multiple different personal devices to communicate simultaneously with the IMD, and this communication privilege can be delegated to further devices and also revoked whenever necessary. This gives the user full control over the access to the communication channel to the IMD.

8.2 Practical realisation of the security manager

The security manager plays an important role in the overall security architecture, as it is the root of trust in the system. From a *security* point of view, it is important that the security manager does not easily get compromised or stolen, and that strong user authentication is enforced on this device (i.e. only the legitimate user can operate the security manager). Moreover, to ensure availability, the security manager should always be available when needed (i.e. when a personal device needs to get granted access to the IMD), wherever the patient is. It should also not be a critical point of failure, meaning that one can securely replace it with a new security manager. From a *usability* point of view, the security manager should be easy to use, have sufficient I/O interfaces (e.g., a keyboard and a display) for user interaction, not introduce significant extra costs, and it should not be a burden for the patient to have access to the security manager (e.g. carrying around a bulky device). From a *deployability* point of view, one should preferably use a personal commercial device that supports standard wireless technologies as a security manager, without requiring any hardware changes to be made to this device. Below we detail possible ways to realise the security manager in practice.

Smartphone-only realisation. When considering the discussion above, the smartphone of the patient would be a good option to realise the concept of the security manager. Indeed, most people always carry their smartphone with them, and therefore it is likely that this device is available when needed (e.g. every time access from a new device to the IMD should be granted). Most modern smartphones now also have hardware protection, such as Trusted Platform Modules (or similar), offering strong security protection for the keys stored on the device. Moreover, since the IMD is implanted in the patient, by default the smartphone and the IMD will be within the communication range most of the time. Therefore, we consider this to be the default option to realise the security manager. However, the main disadvantage of using personal devices is that they can easily get stolen, lost or damaged. While this is mitigated by giving several bootstrap credentials to patients, it is likely that the system needs to be reset, and a new security manager has to be initialised, multiple times. This would, however, hinder the usability of the system.

Alternative realisations. Taking into account the disadvantages mentioned above, one could go one step further and store the cryptographic key of the security manager (or even the multiple bootstrap credentials) in the cloud. There already exist multiple commercial cloud-based key management solutions, where security credentials are stored securely in an online digital vault. Cloud providers are now also starting to offer Trusted Execution Environments (TEE) enabled Infrastructure as a Service (IaaS) solutions to their customers (e.g., Microsoft Azure Confidential [30]), to increase the trustworthiness. An interesting advantage of this approach is that one does not always need to use the same personal device as the security manager. As long as the device is connected to the cloud and operated by the correct user, it could act as the security manager. Obviously, strong user and device authentication should be enforced to ensure that nobody else but the user can get access to the cloud and the keys. This cloud-based approach only requires that the smartphone has online access to the cloud, which should not be a problem. Most hospitals offer Wi-Fi connections and have mobile coverage. Moreover, one should note that access to the cloud-based security manager is only needed when a new personal device needs to be paired with the IMD, which will not happen that often. Also, cloud-based security managers make it easier to delegate access immediately if an emergency takes place, due to its constant availability. Moreover, another advantage of cloud managers is that the bootstrap codes cannot get lost. Due to the interesting security, usability and deployability properties, this cloud version of the security manager is a valuable option to consider.

However, there are also other trade-offs to consider with respect to the realisation of the security manager. For example, not all patients have a smartphone or are willing to use their personal devices for this application. For these patients, one could use a low-cost dedicated commercial device. This could, for example, be an embedded device - or even an altered smartphone - with only one functionality enabled: being a security manager. This device can then be used offline and stored securely when not being used or it can be used as the interface to the cloud-based security manager, similarly to the case where the patient uses the smartphone. However, depending on the exact hardware being used, it might offer weaker protection of the authentication key and hash chain of access tokens that are established with the IMD.

9 RELATED WORK

In Appendix A, we list the design requirements a modern IMD should have, and we evaluate our work with existing work and we visually compare them in Table. 1 and show that the prior state-of-the-art fails to fulfil all these requirements at once. Therefore, our goal is to design a solution that does meet all these requirements. Various solutions have been proposed for establishing a secure wireless channel between an IMD and a personal device. These can be split into two categories: pairing solutions relying on an (additional) external device, and solutions not requiring any additional device.

9.1 Pairing based on an external device

Various countermeasures have been introduced based on the use of an external device to mediate between a device programmer and an IMD. For example, Gollakota et al. proposed an external device known as a “shield” that jams the messages to/from the IMD to prevent others from decoding them, while still being able to successfully decode the messages itself [13]. Nevertheless, the shield offers limited protection since adversaries could bypass it by transmitting messages with more power than those sent by the shield. Furthermore, a multiple-input multiple-output (MIMO) eavesdropper could suppress the jamming signal and recover the data sent by the devices, as shown by Tippenhauer et al. [42]. Xu et al. presented the "IMDGuard", a wearable proxy device that performs an authentication process on the ICD's behalf using the patient's electrocardiography (ECG) signals [45]. However, Rostami et al. found that the IMDGuard is vulnerable to a man-in-the-middle (MiTM) attack [34]. In addition, a common limitation of all these solutions is that they could jam transmissions sent by legitimate devices. In some countries, jammers are illegal and their use can result in large fines. Another disadvantage is that one either needs to use the proxy all the time (as when the shield is used), or every time a new authentication instance takes place (as when IMDGuard is used). In our scheme HAT, however, the security manager does *not* need to be in the vicinity of the IMD at all times.

These above-existing solutions require the users to carry the external device whenever access to the IMD is needed (e.g., in the case of the IMDGuard) or to protect the IMD against attacks (e.g., in the case of the shield). In HAT, one only needs the security manager for access control management, i.e., only when adding or revoking personal devices. Most of the existing solutions do not support efficient revocation. We do not require the external device to be involved in each and every authentication session carried out between the personal device and the IMD. Also, none of the existing solutions uses the external device as a cryptographic token manager, it only uses them to evaluate and authenticate users. In our work, the security manager is responsible for managing all the cryptographic tokens being distributed and can authenticate and revoke access at will. Moreover, none of the existing solutions allows access to the IMD in the case of a medical emergency. In HAT, the medical staff can access the IMD in most cases. In practice, IMD access in an emergency case means that the medical staff requires access to the security manager. If the patient's smartphone was the security manager, one could safely assume that the medical staff could unlock it via fingerprint or facial recognition. Alternatively, if somebody else besides the patient is managing the security manager, then a token can be delivered remotely to the medical staff.

9.2 Pairing without any external device

A simple solution for establishing a key between an IMD and a personal device without relying on an external device would be to print a code either on the patient's skin or on a bracelet worn by the patient. This way, medical staff can obtain the code (from which the IMD's key can be derived) and use it to access the patient's IMD. However, adversaries can easily obtain the code by being close to the patient or by stealing their bracelet. Moreover, tattoos can become unreadable after an accident or refused by patients due to cultural, social or personal reasons [11], while bracelets can be lost or damaged. Finally, as bracelets are visible, they can implicitly reveal the patient's condition, which could lead to discrimination. Prior work has been done to establish a cryptographic key with a personal device through an *audio* (e.g., [15]) or a *vibration* channel (e.g., [1, 20, 38]). The underlying assumption of all these solutions is that only personal devices in close proximity (a few centimetres) to the patient's IMD can receive the audio/vibration signals containing the cryptographic key. Yet, they have proven to be vulnerable to security attacks [2, 14]. Besides, these solutions require extra hardware components in IMDs, which increases their complexity and size. Apart from these, the use of the body's physiological signals (PS) to establish a key between an IMD and personal devices [17, 32, 35] and relying on distance bounding protocols [33] have also been studied. Dodis et al. [12] and Linnartz et al. [19] were the first to propose so-called fuzzy cryptographic primitives to allow two devices that measure slightly different PSs to agree on a common cryptographic key. Unfortunately, several researchers have reported that PSs (i) may not provide sufficient entropy in some circumstances [31], (ii) can be obtained remotely [7, 40] and (iii) are often used along with insecure security protocols [24]. Also, unfortunately, the secure implementation of RF-based distance bounding protocols suitable for a resource-constrained verifier (the IMD) still remains an open research problem today. Moreover, as shown by Sedighpour et al., ultrasound-based distance bounding protocols are vulnerable to wormhole attacks [39]. Finally, another limitation of distance bounding protocols is that they would require hardware changes in both the IMD and the personal device, limiting their deployability. Therefore, none of the proposed pairing solutions that do not rely on an external device meet the practical requirements imposed by the latest generations of IMDs.

10 CONCLUSION

Implantable Medical Devices strongly rely on wireless communication for remote monitoring and reconfiguration. While older generations of IMDs made use of insecure proprietary wireless protocols, current IMDs use standard wireless technologies. The use of standard wireless technologies allows to easily connect commercial personal devices such as smartphones to the IMD, enabling novel use cases and more personalised medical treatments. Nevertheless, the initial bootstrap problem – i.e. how to securely establish a cryptographic key between the IMD and one or more personal devices – had not yet been solved. Key establishment schemes specified in wireless standards are not compatible with IMDs, and security solutions proposed in the literature are either insecure or not realistic to deploy in practice.

This paper presents a secure, practical and easy-to-deploy key establishment solution for the latest generation IMDs that meets all the security and functional requirements. Our solution provides fine-grained and dynamic access control with support for revocation and delegation and enables the realisation of novel healthcare use cases that are envisioned by the medical community. It also gives the user full control and transparency over which personal devices should be authorised to connect to the IMD. Our key establishment solution relies on the concept of a security manager – an external device controlled by the user that evaluates access requests on behalf of the IMD. Our solution builds upon HAT, a secure key establishment protocol that uses hash-chain-based tokens issued by the security manager. Compared with other pairing approaches relying on an external device, our proposed solution is the first one that does not require the external device (i.e., the security manager) to be continuously present during communication with the IMD. Moreover, we demonstrate that our scheme has no significant impact on energy consumption and memory overhead, and can be realised on an MSP430, a common low-cost mid-range microcontroller similar to the one used in commercial IMDs. Moreover, we analysed the security of our protocol using Verifpal.

ACKNOWLEDGEMENTS

This work was supported in part by CyberSecurity Research Flanders with reference number VR20192203, by the FWO SBO project SPITE, by the European Union's Horizon 2020 Research and innovation program under grant agreements No. 826284 (ProTego) and No. 871793 (Accordion), and by VLAIO in the context of the CS-ICON project TRUSTI.

REFERENCES

[1] S.A. Abhishek and N. Saxena. 2016. Vibreaker: Securing Vibrational Pairing with Deliberate Acoustic Noise. In *Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. ACM Press, 103–108.

[2] S A. Anand and N. Saxena. 2017. Coresident Evil: Noisy Vibrational Pairing in the Face of Co-Located Acoustic Eavesdropping. In *Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. ACM Press, 173–183.

[3] M. Bellare and P. Rogaway. 2005. *Introduction to Modern Cryptography.* 207 pages.

[4] Bruno Blanchet. 2016. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends® in Privacy and Security* (2016), 1–135.

[5] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. 2012. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *IEEE Symposium on Security and Privacy (S&P)*. 553–567.

[6] B. Buhrow, P. Riemer, M. Shea, B. Gilbert, and E. Daniel. 2015. lock Cipher Speed and Energy Efficiency Records on the MSP430: System Design Trade-Offs for 16-Bit Embedded Applications. In *Progress in Cryptology - LATINCRYPT*. 104–123.

[7] A. Calleja, P. Peris-Lopez, and J. E. Tapiador. 2015. Electrical Heart Signals can be Monitored from the Moon: Security Implications for IPI-Based Protocols. In *Information Security Theory and Practice*. 36–51.

[8] CareLink [n. d.]. CareLink 2090 Programmer with RemoteControl Technology. https://www.youtube.com/watch?v=Au17Z_cp79g.

[9] D. Coppersmith and M. Jakobsson. 2003. Almost Optimal Hash Sequence Traversal. In *Financial Cryptography (FC)*. 102–119.

[10] T. Denning, A. Borning, B. Friedman, B. T. Gill, T. Kohno, and W. H. Maisel. 2010. Patients, pacemakers, and implantable defibrillators: human values and security for wireless implantable medical devices. In *International Conference on Human Factors in Computing Systems (CHI)*. 917–926.

[11] T. Denning, D. B. Kramer, B. Friedman, M. R. Reynolds, B. Gill, and T. Kohno. 2014. CPS: Beyond Usability: Applying Value Sensitive Design Based Methods to Investigate Domain Characteristics for Security for Implantable Cardiac Devices. In *Annual Computer Security Applications Conference (ACSAC)*. 426–435.

[12] Y. Dodis, L. Reyzin, and A. Smith. 2004. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In *Advances in Cryptology - EUROCRYPT*. 523–540.

[13] S. Gollakota, H. Hassanieh, B. Ransford, D. Katabi, and K. Fu. 2011. They Can Hear Your Heartbeats: Non-Invasive Security for Implantable Medical Devices. In *SIGCOMM Conference*. 2–13.

[14] T. Halevi and N. Saxena. 2010. On Pairing Constrained Wireless Devices Based on Secrecy of Auxiliary Channels: The Case of Acoustic Eavesdropping. In *Conference on Computer and Communications Security (CCS)*. 97–108.

[15] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel. 2008. Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses. In *IEEE Symposium on Security and Privacy (S&P)*. 129–142.

[16] G. Hinterwälder, A. Moradi, Mi. Hutter, P. Schwabe, and C. Paar. 2015. Full-Size High-Security ECC Implementation on MSP430 Microcontrollers. In *LATINCRYPT*. 31–47.

[17] Chunqiang Hu, Xiuzhen Cheng, Fan Zhang, Dengyuan Wu, Xiaofeng Liao, and Dechang Chen. 2013. OPFKA: Secure and efficient Ordered-Physiological-Feature-based key agreement for wireless Body Area Networks. In *2013 Proceedings IEEE INFOCOM*. 2274–2282.

[18] M. Jakobsson. 2002. Fractal hash sequence representation and traversal. In *IEEE International Symposium on Information Theory,*. 437–444.

[19] L. Jean-Paul and T. Pim. 2003. New Shielding Functions to Enhance Privacy and Prevent Misuse of Biometric Templates. In *Audio- and Video-Based Biometric Person Authentication*. 393–402.

[20] Y. Kim, W. S. Lee, V. Raghunathan, N. K. Jha, and A. Raghunathan. 2015. Vibration-based secure side channel for medical devices. In *Design Automation Conference (DAC)*. 1–6.

[21] Nadim Kobeissi, Georgio Nicolas, and Mukesh Tiwari. 2020. Verifpal: Cryptographic Protocol Analysis for the Real World. In *INDOCRYPT*. 151–202.

[22] Silvia Krug and Mattias O'Nils. 2019. Modeling and Comparison of Delay and Energy Cost of IoT Data Transfers. *IEEE Access* 7 (2019), 58654–58675.

[23] C Li, A Raghunathan, and N K. Jha. 2011. Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system. In *International Conference on e-Health Networking, Applications and Services*. 150–156.

[24] E. Marin, E. Argones R.úa, D. Singelée, and B. Preneel. 2019. On the Difficulty of Using Patient's Physiological Signals in Cryptographic Protocols. In *Symposium on Access Control Models and Technologies (SACMAT)*. 113–122.

[25] E. Marin, D. Singelée, F. D. Garcia, T. Chothia, R. Willems, and B. Preneel. 2016. On the (in)Security of the Latest Generation Implantable Cardiac Defibrillators and How to Secure Them. In *Annual Conference on Computer Security Applications (ACSAC)*. 226–236.

[26] E. Marin, D. Singelée, B. Yang, I. Verbauwhede, and B. Preneel. 2016. On the Feasibility of Cryptography for a Wireless Insulin Pump System. In *Conference on Data and Application Security and Privacy (CODASPY)*. 113–120.

[27] E. Marin, D. Singelée, B. Yang, V. Volski, G. A. E. Vandenbosch, B. Nuttin, and B. Preneel. 2018. Securing Wireless Neurostimulators. In *Conference on Data and Application Security and Privacy (CODASPY)*. 287–298.

[28] J.M. McCune, A. Perrig, and M.K. Reiter. 2005. Seeing-is-believing: using camera phones for human-verifiable authentication. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*. 110–124.

[29] Medtronic CareLink [n. d.]. Medtronic CareLink home monitor. https://www.youtube.com/watch?v=K1_7ecUBAJI.

[30] Microsoft. [n. d.]. Introducing Azure confidential computing. https://azure.microsoft.com/en-us/blog/introducing-azure-confidential-computing/.

[31] L. Ortiz Martin, P. Picazo-Sanchez, P. Peris-Lopez, and J. Tapiador. 2018. Heartbeats Do Not Make Good Pseudo-Random Number Generators: An Analysis of the Randomness of Inter-Pulse Intervals. *Entropy* 20 (2018), 94.

[32] C. C.Y. Poon, Yuan-Ting Zhang, and Shu-Di Bao. 2006. A Novel Biometrics Method to Secure Wireless Body Area Sensor Networks for Telemedicine and M-Health. *Comm. Mag.* 44, 4 (Sept. 2006), 73–81.

[33] K. Bonne Rasmussen, C. Castelluccia, T. S. Heydt-Benjamin, and S. Capkun. 2009. Proximity-Based Access Control for Implantable Medical Devices. In *Conference on Computer and Communications Security (CCS)*. 410–419.

[34] M. Rostami, W. Burleson, A. Juels, and F. Koushanfar. 2013. Balancing security and utility in Medical Devices?. In *IEEE Design Automation Conference (DAC)*.

[35] M. Rostami, A. Juels, and F. Koushanfar. 2013. Heart-to-Heart (H2H): Authentication for Implanted Medical Devices. In *Conference on Computer and Communications Security (CCS)*. 1099–1112.

[36] M. Rushanan, A. D. Rubin, D. F. Kune, and C. M. Swanson. 2014. SoK: Security and Privacy in Implantable Medical Devices and Body Area Networks. In *IEEE Symposium on Security and Privacy*. 524–539.

[37] N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. 2006. Secure device pairing based on a visual channel. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*. 6 pp.–313.

[38] N. Saxena, M. B. Uddin, J. Voris, and N. Asokan. 2011. Vibrate-to-unlock: Mobile phone assisted user authentication to multiple personal RFID tags. In *International Conference on Pervasive Computing and Communications (PerCom)*. 181–188.

[39] S. Sedighpour, S. Capkun, S. Ganeriwal, and M. Srivastava. 2005. Implementation of Attacks on Ultrasonic Ranging Systems. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

[40] R. M. Seepers, W. Wang, G. de Haan, I. Sourdis, and C. Strydis. 2018. Attacks on Heartbeat-Based Security Using Remote Photoplethysmography. *IEEE Journal of Biomedical and Health Informatics* 22, 3 (2018), 714–721.

[41] Frank Stajano and Ross Anderson. 1999. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In *International Workshop on Security Protocols*. 172–182.

[42] N. O. Tippenhauer, L. Malisa, A. Ranganathan, and S. Capkun. 2013. On Limitations of Friendly Jamming for Confidentiality. In *IEEE Symposium on Security and Privacy (S&P)*. 160–173.

[43] Erich Wenger, Thomas Unterluggauer, and Mario Werner. [n. d.]. 8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors. In *INDOCRYPT 2013*.

[44] Erich Wenger and Mario Werner. [n. d.]. Evaluating 16-Bit Processors for Elliptic Curve Cryptography. In *CARDIS 2011*. 166–181.

[45] F. Xu, Z. Qin, C. C. Tan, B. Wang, and Q. Li. 2011. IMDGuard: Securing implantable medical devices with the external wearable guardian. In *International Conference on Computer Communications (INFOCOM)*. 1862–1870.

A DESIGN REQUIREMENTS

This section identifies the design requirements for a key establishment protocol for modern IMDs. Inspired by the authentication framework of Bonneau et al. [5], we select and group the design requirements into three categories: Security (S), Usability (U) and

Table 1: Comparison with other relevant works
Design requirements are ● satisfied, ◑ partially-satisfied and ❍ unsatisfied.

Design Requirements		**Tattoos & bracelets** [11]	**Vibrational & audio channels** [1, 15, 20, 38]	**Physiological signals** [17, 32, 35]	**Distance bounding** [33]	**External devices** [13, 45]	**HAT** [this work]
Security	**S1**	❍	❍	❍	❍	❍	●
	S2	❍	❍	❍	❍	●	●
	S3	❍	●	◑	◑	◑	●
	S4	◑	◑	◑	●	◑	●
Usability	**U1**	●	●	●	●	●	●
	U2	❍	◑	❍	❍	◑	●
	U3	❍	◑	◑	◑	◑	●
	U4	●	●	●	●	●	●
	U5	❍	●	●	●	❍	●
	U6	◑	●	●	●	◑	●
Deployability	**D1**	●	❍	❍	❍	◑	●
	D2	●	◑	◑	❍	◑	●
	D3	❍	◑	◑	◑	◑	●
	D4	●	◑	◑	❍	◑	●

Deployability (D). Apart from the solution being cryptographically secure, that is, no known attacks against the key establishment solution exist and the security solution achieves at least a 128-bit security level, the following requirements should also be guaranteed:

Strong cryptographic protection (S1): There should be no known cryptographic attacks against the key establishment solution, and the false acceptance rate should be very low; i.e. the probability that an unauthorised device succeeds in establishing a key with the IMD should be very low. Moreover, the security solution should achieve a 128-bit security level.

Support for dynamic access control (S2): The key establishment and access control scheme should support a dynamic set of devices. Multiple devices can establish a secure connection to the IMD, and this list of devices can change regularly. Besides establishing a key between the IMD and a new personal device, the following functionality should be supported:

Revocation of external devices (S2a): It should be possible to revoke the access of an external device to the IMD. The revoked device should no longer be able to establish a session key with the IMD.

Support for delegation (S2b): It should be possible to give an external device (temporarily) the possibility to establish a secure communication session with the IMD. The user should have full control over which external device is authorised to delegate access to the IMD, and the number of delegations that the external device could grant.

Key-independence (S3): an external device that has established a session key with the IMD should have no knowledge of the session keys established between the IMD and any other external devices. This also includes session keys from past or future protocol instances.

Availability (S4): Availability should be guaranteed at all times; it should never occur that it is technically no longer possible to establish any new secure session key to the IMD. Note that in an IMD setting, availability is absolutely the most important security requirement.

Limited user participation (U1): No actions by the patient should be required to establish a fresh secure session key between the patient's personal device and its IMD. User interaction should be required only when a personal device needs to be added, removed or replaced within the list of devices that are authorised to communicate with the IMD.

End control by the user (U2): While the participation of the patient or the doctor should be limited, they should have full control over which external devices are paired to the IMD. Without user consent, these pairings should not take place.

Easy-to-use and accepted by users (U3): Any action that is required by the user, should be easy to perform for an average user who does not have any technical expertise. Moreover, the acceptability of the solution among users with different social backgrounds is key for its widespread adoption.

Memory-wise effortless (U4): The user should not be required to memorise any passwords or codes, besides those that were already required by the use of their personal device (e.g. to unlock it)[¶].

Nothing to carry (U5): The user should not be required to carry any *additional* device or wearable, besides the (personal) devices that the user is carrying anyhow (e.g., smartphone or smartwatch).

Easy recovery (U6): It should be easy to recover from the loss of any personal device and re-initialise the system. Also, the initial bootstrap procedure should be easy to perform.

Reliability (D1): The security solution should have a low false rejection rate; i.e., the probability that the key establishment between the IMD and an authorised personal device fails should be negligible.

Support for personal devices (D2): The security solution should support the use of personal devices (i.e. *BYOD*), both by the patient or medical staff. These should be commercial, off-the-shelf personal devices. Neither hardware changes nor the use of any expensive equipment should be required. Moreover, it should be possible to replace any of the personal devices, without sacrificing security.

Scalability (D3): The key establishment solution should support at least a limited number of personal devices that can communicate with the IMD. The exact number depends on the specific use case but is expected to be low.

Lightweight (D4): It should be feasible to implement the key establishment solution in the embedded platforms that are currently used by IMDs, taking into account the limited computation power and memory storage. The security solution should not have a strong impact on the energy consumption of the IMD. The solution should also consider the IMD's lack of input/output interfaces.

[¶]Note that relying solely on a password that the patient has to remember, can also hamper availability in some circumstances, e.g., if the patient is unconscious.